Appium okruženje za automatizirano testiranje mobilnih aplikacija

Bekić, Mia

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

APPIUM OKRUŽENJE ZA AUTOMATIZIRANO TESTIRANJE MOBILNIH APLIKACIJA

Diplomski rad

Mia Bekić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 19.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Mia Bekić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1184R, 07.10.2021.
OIB studenta:	08751971953
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	
Sumentor iz tvrtke:	Željko Brdarić
Predsjednik Povjerenstva:	prof. dr. sc. Goran Martinović
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	izv. prof. dr. sc. Tomislav Matić
Naslov diplomskog rada:	Appium okruženje za automatizirano testiranje mobilnih aplikacija
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti dostupne alate i okruženja za automatizirano testiranje mobilnih (iOS/Android) aplikacija. Nakon provedenog istraživanja potrebno je opisati Appium okruženje te ga usporediti s ostalim postojećim rješenjima. U praktičnom dijelu rada potrebno je primijeniti Appium okruženje na odabranoj mobilnoj aplikaciji. Prilikom primjene okruženja potrebno je, osim izrade osnovnog seta testova, opisati Gherkin sintaksu koja se koristi za opis poslovne logike na ljudski čitljivom jeziku te dodatno prikazati poveznice osnovnog seta testova sa Zephyr Toolom kao Test Management alatom. Tema rezervirana za: Mia Bekić Sumentor iz tvrtke: Željko Brdarić (DICE d.o.o)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2023.

Ime i prezime studenta:	Mia Bekić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1184R, 07.10.2021.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Appium okruženje za automatizirano testiranje mobilnih aplikacija**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Svakim danom tehnologija sve više napreduje, a do sada su njenim razvojem nastali brojni uređaji kao što su računala, televizori, mobilni telefoni i drugi. Razvojem spomenutih uređaja dolazi i do razvoja programa za te uređaje. Razvoj programa nije jednostavan proces i on zahtjeva mnogo vremena i truda. Razvojni proces programa još se naziva i razvojnim životnim ciklusom programa. On se sastoji od nekoliko faza od kojih je jedna i testiranje. Testiranje predstavlja vrlo važnu fazu razvojnog životnog ciklusa programa jer se njime nastoji osigurati što veća kvaliteta programa. Testiranjem se provjerava radi li program na ispravan način i ispunjava li postavljene zahtjeve na njega.

Postoje dva načina testiranja, a to su: ručno testiranje i automatizirano testiranje. Ručno testiranje provodi tester na način da samostalno prolazi kroz program što može biti naporno i dugotrajno. Zbog toga se pojavljuje potreba za automatiziranim testiranjem. Automatizirano testiranje zapravo izvodi alat za automatizirano testiranje što značajno olakšava izvođenje testiranja i ubrzava vrijeme njegovog izvođenja. Postoje brojni alati za automatizirano testiranje, a njegov odabir zasniva se na potrebama i ograničenjima projekta te sposobnostima testera. U ovom diplomskom radu naglasak je na Appium alatu za automatizirano testiranje.

Zadatak ovog diplomskog rada je opisati Appium alat, usporediti ga s ostalim postojećim alatima za automatizirano testiranje mobilnih aplikacija te ga primijeniti prilikom automatiziranog testiranja željene mobilne aplikacije. Osim Appiuma, potrebno je opisati i koristiti Cucumber alat koji koristi Gherkin sintaksu te Zephyr alat za upravljanje testiranjem.

Rješavanje zadatka započelo je istraživanjem dostupnih alata za automatizirano testiranje mobilnih aplikacija. Nakon provedenog istraživanja, opisan je Appium alat i uspoređen je s ostalim dostupnim alatima za automatizirano testiranje mobilnih aplikacija. Uslijedilo je proučavanje i opis načina testiranja mobilnih aplikacija te odabir aplikacije za testiranje. Nakon odabira aplikacije za testiranje, istražen i opisan je način testiranja sličnih aplikacija u praksi te je definiran testni plan. Nadalje, dizajnirani su testni slučajevi prilikom čega su korišteni i opisani Zephyr i Cucumber alati. Nakon toga, izrađen je Appium projekt kojim je izvođeno automatizirano testiranje te su neki njegovi dijelovi također i opisani. Zadatak ovog diplomskog rada završen je izvođenjem automatiziranog testiranja korištenjem lokalnog i udaljenog poslužitelja na različitim uređajima te pregledom i usporedbom rezultata testiranja.

Nakon prvog uvodnog poglavlja, u drugom poglavlju objašnjeno je testiranje mobilnih aplikacija, objašnjeni su načini, tipovi i razine testiranja, spomenuta je odabrana aplikacija za testiranje i objašnjena je praksa testiranja sličnih aplikacija te je definiran testni proces za testiranje odabrane aplikacije. U trećem poglavlju se klasificiraju, opisuju i uspoređuju alati za automatizirano testiranje mobilnih aplikacija. Nadalje, u četvrtom poglavlju opisana je odabrana aplikacija za testiranje, izvedeno je testno planiranje i definirana je testna okolina. U petom poglavlju navedeno je što će se automatizirano testirati, objašnjen je način dizajniranja i implementacije testnih slučajeva, objašnjena je izrada projekta za automatizirano testiranje te je provedeno automatizirano testiranje korištenjem lokalnog poslužitelja. Zatim, u šestom poglavlju analizirani su rezultati testiranja, objašnjen je uvoz rezultata testiranja u Zephyr alat, izvedeno je automatizirano testiranje korištenjem udaljenog poslužitelja i uspoređeni su rezultati testiranja korištenjem lokalnog i udaljenog poslužitelja. U posljednjem, sedmom poglavlju, naveden je zaključak rada.

2. TESTIRANJE MOBILNIH APLIKACIJA

U svakodnevnom životu moguće je pronaći razne aplikacije, a mobilne aplikacije su jedne od njih. Mobilne aplikacije mogu biti: nativne aplikacije, hibridne aplikacije i internetske aplikacije.

- Nativne aplikacije – aplikacije izrađene za točno određenu mobilnu platformu korištenjem programskog jezika specifičnog baš za tu platformu. Imaju potpuni pristup svim API-jima i specifičnim bibliotekama za tu platformu kako bi se maksimalno iskoristile ponuđene značajke mobilnog uređaja. [1]
- Hibridne aplikacije – aplikacije koje se sastoje od različitih internetskih tehnologija kao što su JavaScript i HTML. [1]
- Internetske aplikacije – internetske stranice kojima se može pristupiti preko internetskog preglednika mobilnog uređaja. Neovisne su o mobilnoj platformi i optimizirane su za korištenje preko internetskih preglednika mobilnih uređaja. [1]

Testiranje mobilnih aplikacija, bilo da se radi o internetskim, nativnim ili hibridnim aplikacijama, predstavlja jednu od važnih faza životnog ciklusa razvoja aplikacije. Prema [2], testiranje je zapravo proces izvođenja programa ili sustava s namjerom pronalaženja pogrešaka (engl. *errors*) čime testiranje postaje mjerilo kvalitete programa ili sustava. Prema Hetzelu [3], testiranje pomaže u uspostavljanju povjerenja u ispravnost rada sustava, odnosno aplikacije.

Postoje različiti načini testiranja, a to su ručno testiranje i automatizirano testiranje. Ručno testiranje predstavlja testiranje programa bez upotrebe posebnih testnih alata ili skripti pri čemu tester preuzima ulogu krajnjeg korisnika i testira program s ciljem identificiranja neočekivanog ponašanja ili pogrešaka. [4] Automatizirano testiranje predstavlja tehniku koja automatizira proces validacije funkcionalnosti i osigurava da funkcionalnost zadovoljava zahtjeve prije nego što se aplikacija pusti u produkciju. Svrha automatiziranog testiranja je smanjiti broj testnih slučajeva koji se ručno pokreću, ali ne i eliminirati ručno testiranje tako da nije poanta sve automatizirati, već automatizirati samo ono što je potrebno. Prema [1], način na koji bi aplikacija trebala biti testirana, ručno, automatizirano ili kombinacijom oba tipa testiranja, ovisi o aplikaciji. Ukoliko je aplikacija jednostavna, ima ograničene funkcionalnosti ili je dostupna za preuzimanje samo na određeno vrijeme, dovoljno je izvoditi samo ručno testiranje.

Testiranje mobilnih aplikacija može se izvoditi na emulatorima, simulatorima ili stvarnim uređajima. Emulatori su računalne aplikacije koje se ponašaju jednako kao sklopovlje (engl.

hardware) i operacijski sustav stvarnog uređaja, dok su simulatori manje kompleksne aplikacije koje simuliraju samo mali podskup ponašanja uređaja i njegovog sklopovlja. Glavna razlika između simulatora i emulatora je u tome što emulatori nastoje kopirati cijelu unutarnju strukturu mobilnog uređaja, dok simulatori nastoje samo kopirati ponašanje mobilnog uređaja. Bez obzira na postojanje emulatora i simulatora, testiranje mobilnih aplikacija se treba odvijati na stvarnim uređajima kako bi se postiglo uvjerenje u ispravnost rada aplikacija u stvarnom okruženju. [1]

Osim ručnog i automatiziranog testiranja, za testiranje mobilnih aplikacija važno je spomenuti tipove testiranja koji postoje, a to su funkcionalno i nefunkcionalno testiranje. Prema [1], testiranje funkcionalnosti treba biti prva stvar koju će tester odraditi. Funkcionalno testiranje zasniva se na definiranim funkcionalnim zahtjevima i njegova svrha je provjeriti ispunjava li izrađena aplikacija definirane zahtjeve. Funkcionalno testiranje moguće je izvoditi na svakoj razini testiranja, a razine testiranja su: jedinično testiranje, integracijsko testiranje, testiranje sustava i testiranje prihvatljivosti.

- Jedinično testiranje (engl. *unit testing*) – testiranje kojim se traže nedostaci i kojim se provjerava funkcioniranje komponenata koje se mogu zasebno testirati kao što su objekti, klase, moduli i slično. [5]
- Integracijsko testiranje (engl. *integration testing*) – testiranje sučelja između komponenata i interakcije s drugim dijelovima sustava kao na primjer s datotečnim sustavom, operacijskim sustavom i slično. [5]
- Testiranje sustava (engl. *system testing*) – testiranje kompletnog sustava sa svim integriranim komponentama kojemu je cilj verificirati zadovoljava li sustav tehničke i funkcijske zahtjeve. [4]
- Testiranje prihvatljivosti (engl. *acceptance testing*) – testiranje kojim se nastoji utvrditi spremnost sustava za puštanje u produkciju, a glavni cilj mu je utvrditi jesu li zadovoljene korisničke potrebe.

Nefunkcionalno testiranje služi za testiranje nefunkcionalnih svojstava aplikacije kao što su sigurnost, performanse, upotrebljivost i slično.

Odabrana aplikacija za testiranje u sklopu ovog diplomskom radu je zdravstvena aplikacija koja korisniku omogućava vođenje dnevnika o glavoboljama. Radi se o nativnoj iOS mobilnoj aplikaciji koja sadrži brojne funkcionalnosti. Detaljan opis aplikacije nalazi se u poglavlju 4.

2.1. Pregled testiranja zdravstvenih aplikacija

Kako bi se bilo koja aplikacija, pa tako i zdravstvena aplikacija, testirala na ispravan i kvalitetan način, potrebno je definirati testni proces. Općenito, testni proces prilikom testiranja zdravstvene aplikacije, prema [6], treba izgledati kao što je prikazano na slici 2.1. Naravno, važno je naglasiti da testni proces neće uvijek biti isti nego će se prilagođavati shodno potrebama projekta. Prilikom testiranja zdravstvenih aplikacija, kombinira se ručno i automatizirano testiranje.

Slika 2.1. *Testni proces testiranja zdravstvenih aplikacija.*

Kao **prvi korak** testnog procesa izdvaja se planiranje testiranja. Svrha ovog koraka jest definirati:

- aplikaciju/sustav koji će se testirati
- testnu strategiju
- testno okruženje
- ciljeve testiranja
- područje i ograničenja testiranja
- izvor testnih podataka
- raspored testiranja
- testne detalje za svaku razvojnu fazu. [6]

Drugi korak testnog procesa predstavlja testiranje zahtjeva. Svrha ovog koraka jest analizirati sve definirane zahtjeve (provjeriti njihovu cjelovitost, ispravnost, konzistentnost, mogućnost testiranja) i potencijalne rizike kako bi se predvidjeli problemi koji se mogu pojaviti. [6]

Treći korak testnog procesa je dizajniranje testnih slučajeva na temelju definiranih zahtjeva na aplikaciju. Testni slučajevi su, prema [2], specifični setovi testnih podataka zajedno s očekivanim rezultatima za određeni testni cilj kao što je provjera usklađenosti programske značajke s definiranim zahtjevima.

Četvrti korak testnog procesa je funkcionalno i nefunkcionalno testiranje. Svrha ovog koraka je provjeriti učinkovitost funkcionalnosti, pronaći nedostatke i prijaviti ih razvojnom timu te provjeriti performanse, sigurnost, korisnost i druge aspekte aplikacije. Postoji puno vrsta nefunkcionalnih testiranja, a ona koja se izdvajaju kao najčešće izvođena su: testiranje performansi, testiranje kompatibilnosti, testiranje korisnosti, testiranje usklađenosti i testiranje sigurnosti. [6]

- Testiranje performansi (engl. *performance testing*) – testiranje kojim se nastoji utvrditi kako aplikacija radi s gledišta odziva i stabilnosti pod određenim opterećenjem. [7]
- Testiranje kompatibilnosti (engl. *compatibility testing*) – testiranje kojim se provjerava radi li aplikacija na ispravan način na različitim uređajima, iOS/Android verzijama i slično. [7]
- Testiranje korisnosti (engl. *usability testing*) – testiranje kojim se nastoji utvrditi je li aplikacija prilagođena korisniku i je li jednostavna za korištenje. [7]
- Testiranje usklađenosti (engl. *compliance testing*) – testiranje kojim se utvrđuje poštuje li aplikacija unutarnje/vanjske standarde/zahtjeve; u ovom slučaju odgovarajuće državne zdravstvene propise. [7]
- Testiranje sigurnosti (engl. *security testing*) – testiranje kojim se nastoji otkriti ranjivost sustava i utvrditi da su podaci zaštićeni od mogućih uljeza. [7]

Peti korak testnog procesa je izvještavanje. Izvještaji su potrebni kako bi postojao uvid u sve testne aktivnosti i krajnje testne rezultate. Oni ukazuju na spremnost aplikacije za puštanje u produkciju. [6]

Osim navedenih koraka, potrebno je izdvojiti još i regresijsko testiranje, odnosno testiranje koje nastupa nakon što se učine bilo kakve izmjene u izvornom kodu koje mogu imati utjecaj na rad funkcionalnosti. Služi za verifikaciju ispravnosti rada određenih funkcionalnosti nakon odrađenih promjena u izvornom kodu. Zahtjeva odabir već izvođenih testnih slučajeva koji će se ponovno izvoditi. Također, tijekom regresijskog testiranja, učestala je pojava korištenja automatiziranog testiranja kako bi se testiranje ubrzalo.

U ovom diplomskom radu, testni proces je slično definiran. Sastoji se od testnog planiranja, testiranja zahtjeva, dizajniranja testnih slučajeva i njihovog implementiranja, funkcionalnog testiranja i izvještavanja. Testni proces ovog diplomskog rada ne uključuje regresijsko testiranje jer se u sklopu ovog diplomskog rada nisu radile izmjene na izvornom kodu zbog kojih bi bilo potrebno izvoditi regresijsko testiranje. Fokus je na funkcionalnom testiranju, no izvedenim

funkcionalnim testiranjem provjerila se ispravnost rada funkcionalnosti na različitim uređajima i različitim operacijskim sustavima čime se također izvelo nefunkcionalno testiranje, odnosno testiranje kompatibilnosti. Odabranu aplikaciju za testiranje bi, u praksi, bilo dobro ručno i automatizirano testirati, no kako je područje ovog diplomskog rada automatizirano testiranje, ovim radom pokriveno je samo automatizirano testiranje.

3. ALATI ZA AUTOMATIZIRANO TESTIRANJE MOBILNIH APLIKACIJA

Alati za automatizirano testiranje su programi dizajnirani za validaciju i verifikaciju nefunkcionalnih i funkcionalnih zahtjeva pomoću skripti. Na temelju potreba i ograničenja projekta te znanja/sposobnosti testera, potrebno je odabrati alat za automatizirano testiranje. S obzirom da se u ovom diplomskom radu radi o testiranju mobilne aplikacije, naglasak je na alatima za automatizirano testiranje mobilnih aplikacija. Alati se mogu klasificirati prema brojnim karakteristikama, a jedna od njih je za koju platformu je namijenjen. Na temelju navedenog, alati se dijele na:

1. **Android alate** – na primjer: Espresso, Robotium
2. **iOS alate** – na primjer: XCTest
3. **višeplatformske alate** (iOS i Android) – na primjer: Appium, Katalon Studio, TestProject, Detox.

S obzirom da je u ovom diplomskom radu naglasak na Appium alatu, u nastavku je isti detaljno opisan. Također, ukratko su opisani i neki od ostalih predstavnika pojedine grupe alata kao Espresso, Robotium, XCTest i Katalon Studio, s kojima je Appium uspoređen u poglavlju 3.6.

3.1. Espresso

Espresso je alat otvorenog koda za automatizirano testiranje nativnih i hibridnih Android mobilnih aplikacija kojeg je razvio Google. Pomaže programerima u pisanju automatiziranih testnih slučajeva za testiranje korisničkog sučelja. Testne skripte mogu se pisati u Java ili Kotlin programskom jeziku. Iznimno je brz prilikom izvođenja testova, besplatan je za korištenje i može upravljati automatskom sinkronizacijom aplikacija i testova. Ciljana grupa korisnika su programeri, ali često ga koriste i testeri kojima omogućuje testiranje fragmenata i zasebnih komponenata tijekom razvojnog ciklusa. [8] [9] [10] [11]

3.2. Robotium

Robotium je alat otvorenog koda za automatizirano testiranje nativnih i hibridnih Android aplikacija. Za pisanje testnih skripti koristi se Java programski jezik. Omogućava testiranje korisničkog sučelja, pisanje funkcionalnih testova, sustavnih testova i testova prihvatljivost.

Testiranje se može izvoditi na stvarnim uređajima i emulatorima te ih je moguće izvoditi samo na jednom uređaju istovremeno. Brz je prilikom izvođenja testova i podržava integraciju u cjevovod kontinuirana integracija/kontinuirana implementacija (engl. *Continuous Integration/Continuous Deployment, CI/CD*). [1] [12] [13]

3.3. XCTest

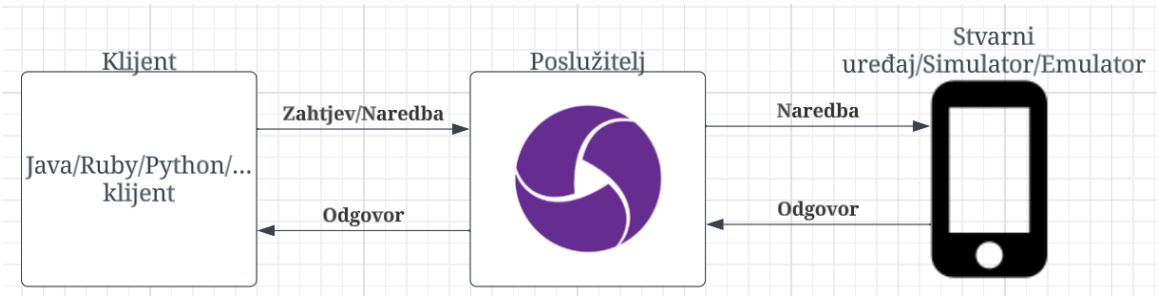
XCTest je alat otvorenog koda za automatizirano testiranje iOS mobilnih aplikacija kojeg je razvio Apple. Besplatan je za korištenje, ugrađen je u razvojno okruženje Xcode te podržava integraciju u CI/CD cjevovod. Koristi se za jedinično testiranje, testiranje performansi i testiranje korisničkog sučelja. Za pisanje testnih skripti koristi se Swift ili Objective-C programski jezik. [14] [15]

3.4. Katalon Studio

Katalon Studio je alat koji omogućava pisanje testnih skripti za internetske, API, mobilne i računalne aplikacije. Dostupan je u besplatnoj i komercijalnoj verziji, podržava Java i Groovy programske jezike za pisanje testnih skripti i ima mogućnost jednostavne integracije u CI/CD cjevovod. Jednostavan je za korištenje jer zahtjeva nisku razinu programiranja, odnosno omogućava razvijanje programske podrške korištenjem već gotovih funkcijskih blokova. [13] [16]

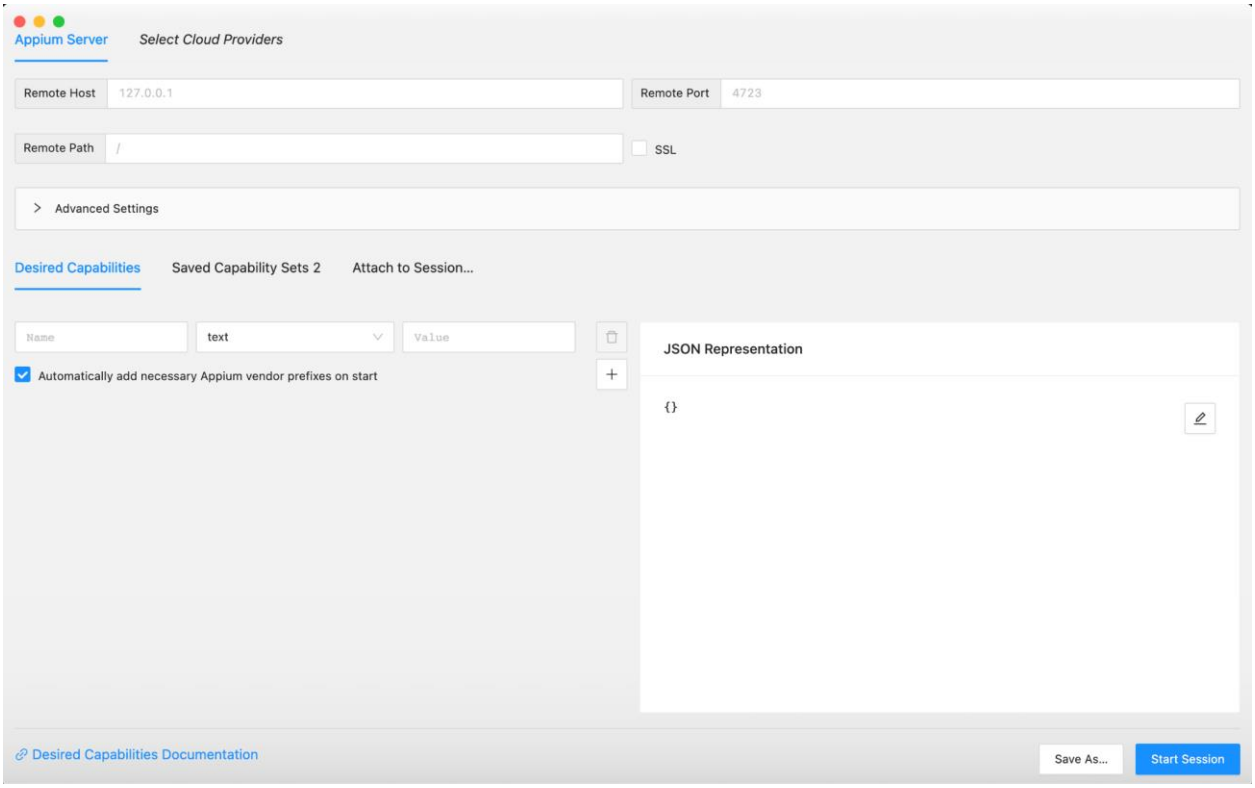
3.5. Appium

Appium je višeplatformski alat otvorenog koda za automatizirano testiranje nativnih, hibridnih, internetskih, TV i stolnih aplikacija. Besplatan je za korištenje te podržava testiranje na simulatorima, emulatorima i stvarnim uređajima. Moguće ga je koristiti na macOS, Windows i Linux operacijskim sustavima. Omogućava pisanje testnih skripti pomoću brojnih programskih jezika kao što su Java, Ruby, Python, JavaScript, C# i drugi te implementira klijent-poslužitelj arhitekturu. [17] [18] U navedenoj klijent-poslužitelj arhitekturi, klijent je odgovoran za slanje naredbi poslužitelju preko mreže (engl. *network*) i primanje njegovih odgovora kao rezultat, dok je poslužitelj povezan s uređajem na kojem se izvodi testiranje te je on zaslužan za izvođenje automatizacije. Poslužitelj je zapravo sam Appium zajedno s korištenim driverom/driverima i dodacima (engl. *plugins*) za automatizaciju. Appium poslužitelj je HTTP poslužitelj i on mora biti pokrenut dokle god se automatizacija želi izvoditi. [18] [19] Appium klijent odabire se na temelju programskog jezika u kojem se skripte žele pisati pa tako postoje *Appium Java client, Appium Python client* i drugi. Prema [18], Appium klijent i poslužitelj ne moraju biti pokrenuti na istom računalu. Na slici 3.1. moguće je vidjeti opisanu arhitekturu.



Slika 3.1. *Appium arhitektura.*

Objavom Appium 2.0 verzije, Appium Desktop, grafičko sučelje za Appium poslužitelj pomoću kojeg se poslužitelj pokretao/zaustavljao i slično, je zastarjelo (engl. *deprecated*). Od tada se, od Appium korisnika, zahtjeva komandno pokretanje poslužitelja. [20] Kako bi se poslužitelj komandno pokrenuo, u terminal je potrebno upisati *appium* ili *appium server* nakon čega se poslužitelj pokreće s predefiniranim (engl. *default*) argumentima. Sve argumente moguće je modificirati, a neki od njih su: *address*, *port*, *callback-port*, *base-path*, *log* i drugi. [21] Također, Appium Desktop je imao ugrađeno korisničko sučelje za pregled i interakciju s elementima aplikacije. Njegovim zastarijevanjem pojavio se Appium Inspector, GUI alat za mobilne i druge aplikacije, kojeg pokreće Appium poslužitelj. Appium Inspector je zapravo Appium klijent s korisničkim sučeljem. Nakon pokretanja Appium Inspectora, prikazuje se sučelje (Slika 3.2.) u kojem je potrebno definirati koji Appium poslužitelj se želi koristiti. Također, u istom sučelju, potrebno je definirati i željene mogućnosti (engl. *capabilities*) sesije koja se nastoji pokrenuti. [22]



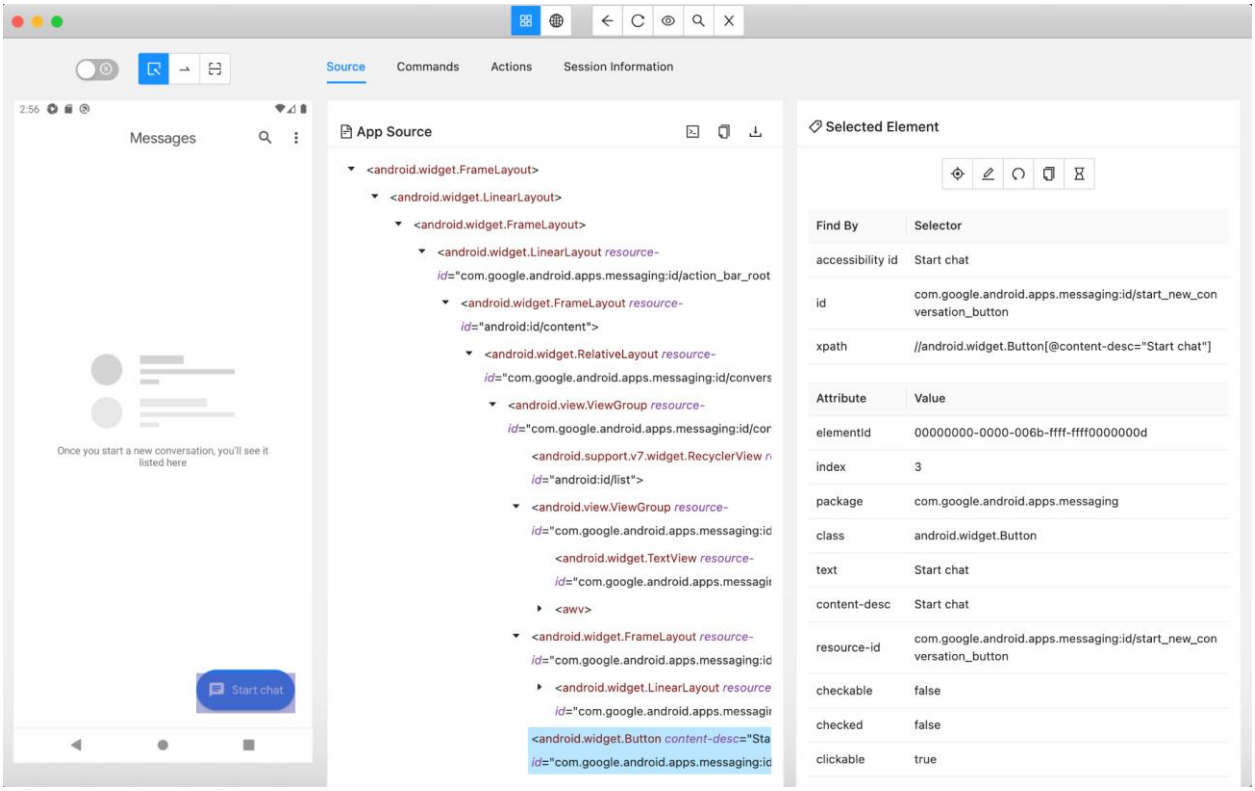
Slika 3.2. *Appium Inspector sučelje za definiranje poslužitelja i željenih mogućnosti sesije.*

Dostupne mogućnosti ovise o driveru koji se koristi, a neke mogućnosti su:

- *platformName* – naziv platforme za koju je izrađena aplikacija
- *appium:automationName* – naziv Appium drivera koji će se koristiti
- *appium:app* – putanja do aplikacije
- *appium:deviceName* – naziv uređaja za automatizaciju
- *appium:platformVersion* – verzija platforme

i druge [23].

Nakon uspješne izrade sesije, dobiva se sučelje u kojem je moguće pregledavati hijerarhiju korisničkog sučelja aplikacije, ostvariti interakciju s elementima, pregledati njihove atribute, izraditi testnu skriptu snimanjem interakcije s elementima i drugo. Navedeno sučelje moguće je vidjeti na slici 3.3.



Slika 3.3. *Appium Inspector sučelje nakon pokretanja sesije i klika na određeni element.*

3.6. Usporedba Appiuma s drugim postojećim alatima za automatizirano testiranje mobilnih aplikacija

Alati koji su odabrani za usporedbu s Appium alatom, već su ranije ukratko opisani u ranijim poglavljima, a to su: Espresso, Robotium, Katalon Studio i XCTest. Iz spomenutih opisa već su se mogle uočiti neke sličnosti i razlike među alatima, no u ovom poglavlju prikazana je tablica 3.1. u kojoj su, na jednom mjestu, prikazane neke sličnosti i razlike navedenih alata.

Tablica 3.1. *Usporedba izdvojenih alata za automatizirano testiranje mobilnih aplikacija.*

	Appium	**Katalon Studio**	**Espresso**	**Robotium**	**XCTest**
Podržane mobilne platforme	iOS, Android	Android, iOS	Android	Android	iOS
Zahtjeva plaćanje	Ne	Ovisi	Ne	Ne	Ne
Podržani jezici	Java, Ruby, Python, C#, JavaScript, ...	Java, Groovy	Java, Kotlin	Java	Swift, Objectiv-C

Podržani testni uređaji	Stvarni uređaji, simulatori, emulatori	Stvarni uređaji, simulatori, emulatori	Stvarni uređaji, emulatori	Stvarni uređaji, emulatori	Stvarni uređaji, simulatori
Postavljanja i konfiguracije	Zahtjeva kodiranje, složeno	Jednostavno	Zahtjeva kodiranje	Zahtjeva kodiranje	Zahtjeva kodiranje
Testiranja koja se izvode	Funkcionalno testiranje, testiranje korisničkog sučelja, testiranje prihvatljivosti	Funkcionalno testiranje	Funkcionalno testiranje, testiranje korisničkog sučelja, testiranje prihvatljivosti	Funkcionalno testiranje, testiranje sustava, testiranje prihvatljivosti	Testiranje jedinica, testiranje performansi, testiranje korisničkog sučelja, funkcionalno testiranje

Svi alati imaju svoju svrhu, prednosti i nedostatke te će s obzirom na potrebe određenog projekta neki od njih bili bolji i prikladniji za primjenu od nekog drugog alata.

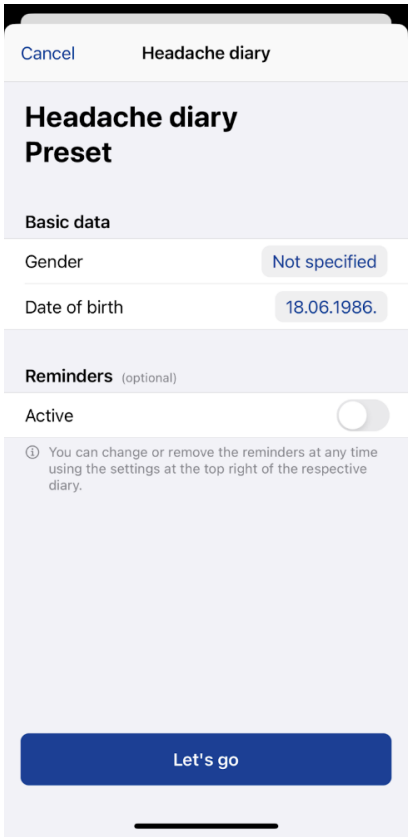
4. PLANIRANJE TESTIRANJA I DEFINIRANJE TESTNE OKOLINE

U ovom poglavlju opisana je odabrana aplikacija za testiranje u sklopu ovog diplomskog rada. Opisano je provedeno planiranje testiranja, definirana je testna okolina i opisani su alati korišteni za dizajniranje, izvođenje i evidenciju automatiziranih testova i njihovih rezultata.

4.1. Zdravstvena aplikacija za vođenje dnevnika o glavoboljama

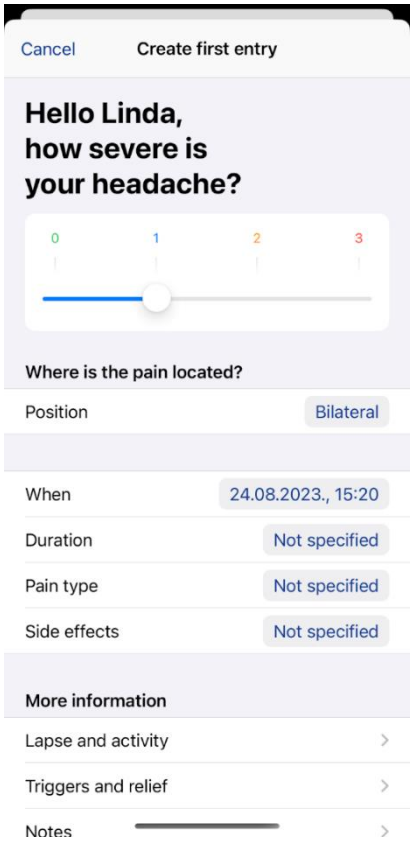
Odabrana zdravstvena aplikacija za testiranje je aplikacija koja predstavlja dnevnik u kojem korisnik može evidentirati i pregledavati podatke o svojim glavoboljama. Pristup dokumentaciji i definiranim zahtjevima postoji, no pristup izvornom kodu aplikacije ne postoji. Aplikacija se sastoji od funkcionalnosti: izrade dnevnika, izrade unosa glavobolje, uređenja unosa glavobolje, uređenja podsjetnika za unos podataka o glavobolji, pregleda unesenih glavobolja, pregleda specifičnog unosa glavobolje, brisanja unosa o glavobolji, pregleda osvojenih i neosvojenih nagrada, arhiviranja i reaktiviranja dnevnika te uređenja korisničkih podataka. Važno je napomenuti da aplikacija nije javno dostupna stoga zbog autorskih prava nije dopušteno prikazati sve zaslone aplikacije i detaljno opisati sve funkcionalnosti.

Funkcionalnost izrade dnevnika zahtjeva definiranje korisničkih podataka, spola i datuma rođenja, pri čemu je dopušten odabir raznih spolova i datuma rođenja do 100 godina od trenutnog datuma. Prilikom izrade dnevnika, moguće je postaviti podsjetnik za unos glavobolje pritiskom na gumb za prebacivanje (engl. *toggle button*) pri čemu je moguće definirati vrijeme i učestalost slanja podsjetnika. Također, u svakom trenutku moguće je odustati od izrade dnevnika. Na slici 4.1. moguće je vidjeti izgled zaslona prilikom izrade dnevnika.



Slika 4.1. *Zaslon aplikacije prilikom izrade dnevnika.*

Funkcionalnost izrade unosa glavobolje moguće je izvesti tek nakon izrade dnevnika. Prilikom izrade unosa potrebno je unijeti podatak o jačini glavobolje. Na temelju tog podatka, koji može imati vrijednost od 0 do 3, određeno je koji podaci se još trebaju ili mogu unijeti, kao što su pozicija boli, datum i vrijeme nastupanja glavobolje, trajanje, tip boli, nuspojave, bilješke i drugi. U svakom trenutku moguće je odustati od izrade unosa glavobolje. Na slici 4.2. prikazan je izgled aplikacije nakon pokretanja funkcionalnosti unosa.



Slika 4.2. *Zaslon aplikacije prilikom izrade unosa glavobolje jačine 1.*

Funkcionalnost uređenja unosa glavobolje zapravo je slična funkcionalnosti izrade unosa. Razlika je u tome što za uređenje unosa mora postojati unos koji će se urediti i što se, prilikom uređenja unosa, unos može obrisati. Također, spremanje promjena je onemogućeno ukoliko promjene nisu učinjene ili obvezni podaci nisu uneseni te je u svakom trenutku moguće odustati od uređenja unosa glavobolje.

Funkcionalnost uređenja podsjetnika za unos podataka o glavobolji je funkcionalnost u kojoj je moguće postaviti ili ukloniti već postavljeni podsjetnik. Kao i kod uređenja unosa glavobolje, spremanje promjena je onemogućeno ukoliko promjene nisu učinjene i u svakom trenutku je moguće odustati od uređivanja podsjetnika.

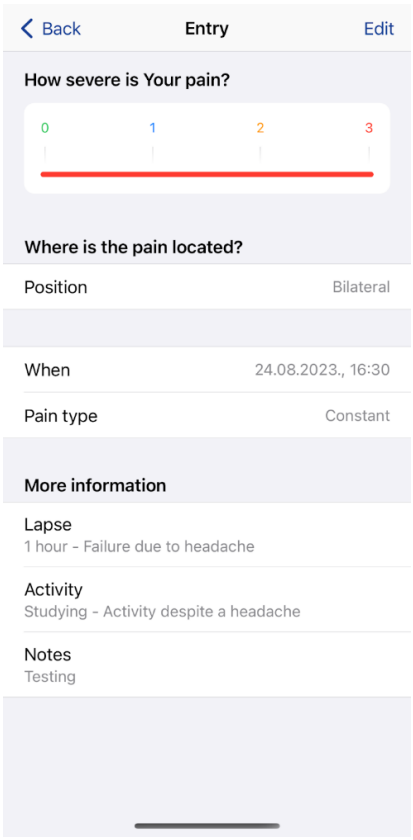
Funkcionalnost pregleda unesenih glavobolja je funkcionalnost koja korisniku omogućava pregled, do sada, unesenih glavobolja.

Funkcionalnost uređenja korisničkih podataka je funkcionalnost koja korisniku omogućava uređenje postavljenog spola i datuma rođenja. Kao i kod ostalih funkcionalnosti uređenja,

spremanje promjena je onemogućeno ukoliko promjene nisu učinjene te je u svakom trenutku moguće odustati od uređenja korisničkih podataka.

Dnevnik posjeduje nagrade koje korisnik može osvojiti izvođenjem određenih akcija pa se tako pojavila i funkcionalnost pregleda osvojenih i neosvojenih nagrada.

Funkcionalnost pregleda specifičnog unosa glavobolje omogućava prikaz svih unesenih podataka prilikom izrade unosa glavobolje (Slika 4.3.). Podaci koji nisu uneseni prilikom izrade unosa glavobolje, a mogli su biti, ne smiju biti prikazani prilikom pregleda specifičnog unosa glavobolje.



Slika 4.3. *Zaslon aplikacije nakon odabira unosa za prikaz.*

4.2. Planiranje testiranja

Planiranje testiranja predstavlja prvi korak u definiranom testnom procesu ovog diplomskog rada. Planiranje je značajno jer se njime određuju mnoge stavke kao: što se testira, na koji način će se testiranje izvoditi, koji su rizici, potrebni testni resursi (ljudi, testno okruženje, uređaji i slično), kada se izvodi testiranje, raspored izvođenja pojedinih aktivnosti i slično. Tijekom planiranja testiranja vrlo je važno razumjeti ciljeve, želje kupaca i osobe koja je zatražila izradu programa

(engl. *stakeholder*) te razumjeti cilj samog projekta. Navedeno je važno jer razumijevanje svega toga pomaže u donošenju ispravnih odluka prilikom planiranja testiranja. [5]

U sklopu ovog diplomskog rada, kao što je već ranije navedeno, izvodi se testiranje iOS mobilne aplikacije za vođenje dnevnika o glavoboljama. Odlučeno je izvesti funkcionalno testiranje navedene aplikacije korištenjem automatiziranog testiranja s ciljem provjere ispravnog rada funkcionalnosti. Za automatizirano testiranje odabrane su točno određene funkcionalnosti i njihovi zahtjevi, o čemu je više napisano u poglavlju 5. Testiranje se izvodi nakon što se određene funkcionalnosti, koje se testiraju, implementiraju u aplikaciju. Za potrebe ovog rada, nije bilo nužno definirati raspored izvođenja pojedinih aktivnosti. Kao rizik testiranja izdvaja se mogućnost nepostojanja potrebnog testerskog znanja za izvođenje svih testnih aktivnosti. Od potrebnih testnih resursa izdvajaju se:

- jedan tester koji će izvoditi sve testne aktivnosti
- internetski pristup
- računala na kojima će se izvoditi testne aktivnosti
- alati potrebni za izvođenje automatiziranog testiranja.

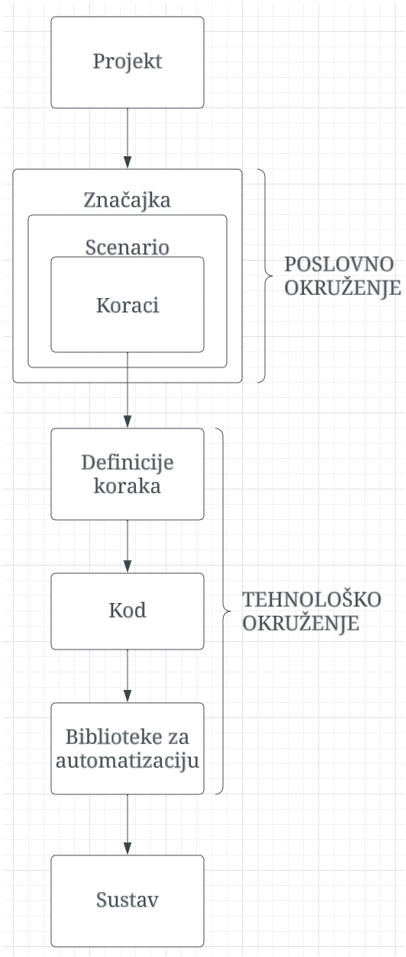
Također, u sklopu ovog diplomskog rada, automatizirano testiranje se izvelo korištenjem Appium poslužitelja pokrenutog lokalno i udaljeno na drugom računalu te se izvršila usporedba potrebnog vremena za izvođenje automatiziranog testiranja. Važno je napomenuti kako su, prilikom izvođenja automatiziranog testiranja korištenjem udaljenog poslužitelja, računalo na kojem je pokrenut udaljeni poslužitelj i računalo na kojem je pokrenut klijent povezani na istu lokalnu mrežu.

4.3. Testno okruženje

Za izvođenje svih testnih aktivnosti korišteno je prijenosno računalo MacBook Pro s M1 procesorom i RAM-om od 16GB te prijenosno računalo Asus VivoBook s procesorom Intel Core i5 i RAM-om od 8GB. Za izvođenje automatiziranog testiranje korišteno je razvojno okruženje IntelliJ IDE, alat Appium, Cucumber i TestNG, a testiranje je izvedeno na 3 različita simulatora: iPhone 8 (iOS 16.1), iPhone 11 (iOS 16.2), iPhone 12 Pro (iOS 16.4). Za dizajniranje, evidenciju automatiziranih testova i rezultata automatiziranog testiranja korišten je Zephyr Squad alat koji dolazi kao dodatak za Jira Software alat, programsku podršku za upravljanje projektima. Za implementaciju automatiziranih testova korišten je Java programski jezik.

4.3.1. Cucumber

Cucumber je alat koji podržava razvoj vođen ponašanjem (engl. *Behavior-Driven Development, BDD*). BDD naglašava razvoj značajki temeljenih na korisničkoj priči. BDD pruža zajednički jezik, baziran na jednostavnim i strukturiranim rečenicama, koji olakšava komunikaciju između članova projektnog tima i osobe koja zahtijeva izradu projekta. [24] [25] Cucumber čita izvršne specifikacije, napisane običnim tekstom u dokumentima zvanim značajke (engl. *features*), i potvrđuje radi li programska podrška ono što specifikacije zahtijevaju od nje. Specifikacije se sastoje od više scenarija ili primjera. Svaki scenarij je lista koraka koje Cucumber mora proći, a kako bi Cucumber mogao razumjeti scenarije, oni moraju poštivati osnovna sintaksna pravila koja se nazivaju Gherkin. Definicije koraka (engl. *step definitions*) povezuju Gherkin korake s programskim kodom, odnosno povezuju specifikaciju s implementacijom. [25] Cucumber ne dolazi s bibliotekom za potvrđivanje (engl. *assertion*) stoga je potrebno koristiti metode za potvrđivanje iz drugih testnih alata kao što je TestNG. Na slici 4.4., moguće je vidjeti Cucumber arhitekturu prikazanu u [26].



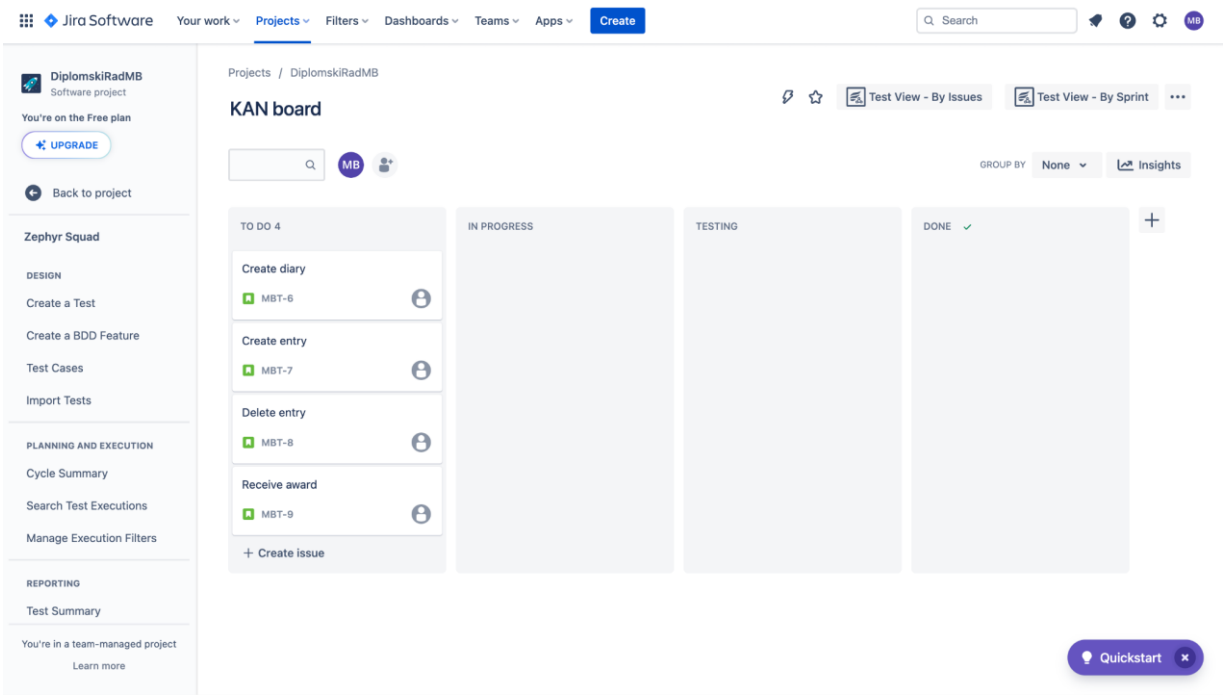
Slika 4.4. *Cucumber arhitektura.*

4.3.2. Zephyr Squad i Jira Software

Zephyr Squad je alat za upravljanje testiranjem koji omogućava dizajniranje, kloniranje, uvoz, izvoz, izvođenje i automatizaciju testova te praćenje izvođenja testnih aktivnosti. Zephyr Squad je komercijalan alat, no nudi besplatno probno razdoblje od mjesec dana nakon čega je plaćanje obavezno i ovisi o odabranoj opciji domaćina (engl. *host option*) i broju korisnika. Omogućava dizajniranje testnih slučajeva u obliku korak po korak (engl. *Step-by-Step*) i u obliku BDD značajki, odnosno u Gherkin sintaksi. Moguće ga je integrirati s CI/CD alatima. Također, prema [27], omogućava uvoz rezultata iz korištenih okvira za automatizirano testiranje (Cucumber, Selenium, JUnit, TestNG, SoapUI, UFT, EggPlant i Tricentis Tosca) izravno u izvođeni testni ciklus. Dolazi kao aplikacija koja se može dodati u Jira Software alat. Jira Software je, prema [28], alat za upravljanje projektima kojeg koriste timovi za planiranje, praćenje, izdavanje i podržavanje programske podrške. Prema tome, Jira Software se sastoji od projekata koji se, prema [29], mogu gledati kao kontejneri korišteni za organizaciju i praćenje zadataka ili problema (engl. *issues*) unutar cijelog tima. Svaki projekt ima svoju ploču (engl. *board*) koja prikazuje probleme u stupcima. Svaki stupac, prema [30], predstavlja jedan korak u radnom tijeku (engl. *workflow*) tima za dovođenje posla do završetka. Problemi u Jira Software alatu mogu bit različitog tipa, kao na primjer:

- *epic* – predstavlja kolekciju problema
- *story* – predstavlja zahtjev prikazan iz perspektive korisnika
- *bug* – predstavlja problem koji treba biti popravljen

i drugi. [31] Na slici 4.5. moguće je vidjeti Jira Software alat, odnosno Jira ploču s četiri izrađena *story* problema te neke od Zephyr Squad funkcionalnosti prikazane u lijevoj navigacijskoj traci.



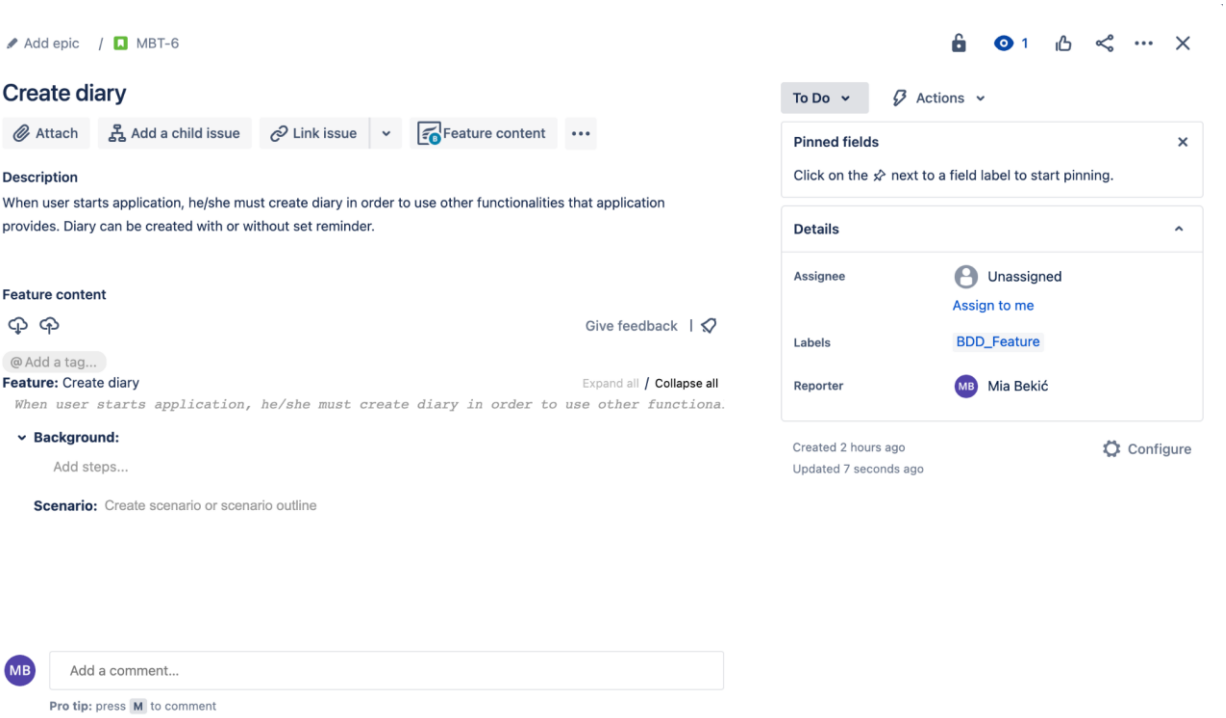
Slika 4.5. *Prikaz ploče s problemima i nekih Zephyr Squad funkcionalnosti.*

5. DIZAJNIRANJE TESTNIH SLUČAJEVA I PROVOĐENJE TESTIRANJA

Nakon planiranja testiranja odrađena je analiza zahtjeva na temelju koje je odlučeno što će se sve automatizirano testirati. Odlučeno je automatizirano testirati:

1. izradu dnevnika s postavljenim podsjetnikom
2. izradu dnevnika bez postavljenog podsjetnika
3. izradu unosa glavobolje s unesenim različitim podacima
4. mogućnost unosa bilješke duljine maksimalno 1000 znakova prilikom izrade unosa glavobolje
5. dobivanje nagrada
6. brisanje unosa glavobolje.

Kao što je već ranije navedeno, Zephyr Squad alat je korišten prilikom dizajniranja testnih slučajeva. S obzirom da se u ovom radu koristi Cucumber, u Zephyr Squad alatu izrađeni su testni slučajevi u obliku BDD značajki. Kako bi BDD značajka mogla biti izrađena, *story* problem mora postojati i mora imati oznaku (engl. *label*) *BDD_Feature*. U *story* problem, koji ima navedenu oznaku, moguće je dodati sadržaj značajke (engl. *feature content*) koji sadrži Gherkin korake. Navedeno je moguće vidjeti na slici 5.1. Za potrebe ovog diplomskog rada izrađeno je 4 *story* problema: *Create diary*, *Create entry*, *Deleting entry*, *Receive award*.



Slika 5.1. *Story problem sa sadržajem značajke.*

5.1. Gherkin

Gherkin je skup gramatičkih pravila koji običan tekst čine dovoljno strukturiranim da ga Cucumber može razumjeti. On koristi skup ključnih riječi kako bi dao značenje i strukturu specifikacijama, a svaka ključna riječ je prevedena na više od 70 govornih jezika i potpuno ih je validno koristiti prilikom izrade Gherkin dokumenata. Gherkin dokumenti koriste *.feature* datotečnu ekstenziju. [25] [32] [33] Neke ključne riječi su [32]: *Feature*, *Scenario*, *Scenario Outline*, *Examples*, *Given*, *Background*, *When*, *Then*, *And* i druge.

- *Feature* – ključna riječ koja uvijek mora biti prva korištena primarna ključna riječ u Gherkin dokumentu, a njena svrha je grupirati povezane scenarije i dati opis značajke na visokoj razini. [32]
- *Scenario* – predstavlja konkretan primjer koji dočarava poslovno pravilo i sastoji se od više koraka. [32]
- *Scenario Outline* – ključna riječ koja se koristi za pokretanje istih scenarija, s različitim vrijednosti, više puta. Uz nju je obavezno koristiti ključnu riječ *Examples*. [32]

- *Examples* – ključna riječ koja se koristi za označavanje sekcije u kojoj se navode vrijednosti koje se mijenjaju prilikom izvođenja scenarija (za svaki novi red u navedenoj sekciji, izvodi se jedan scenarij). [32]
- *Given* – ključna riječ kojom se označava korak koji predstavlja početno stanje sustava, preduvjet koji mora biti ispunjen kako bi se daljnji koraci mogli izvesti. [32]
- *Background* – ključna riječ koja se koristi za označavanje sekcije u kojoj se navode *Given* koraci koji se ponavljaju u svakom scenariju određene značajke. [32]
- *When* – ključna riječ koja označava korak koji predstavlja akciju koju je potrebno izvesti. [32]
- *Then* – predstavlja ključnu riječ koja označava korak koji predstavlja očekivani rezultat. [32]
- *And* – ključna riječ koju je moguće koristiti kada postoji više *Given/When/Then* koraka jedan iza drugog. Njenim korištenjem scenarij postaje fluidniji. [32]

5.2. Primjer dizajniranja testnih slučajeva

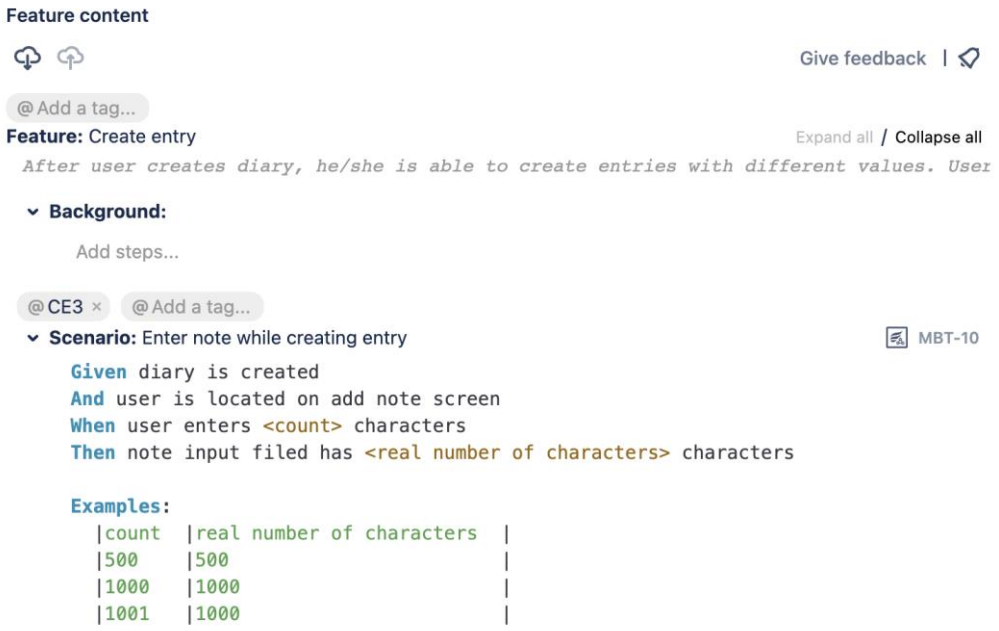
Za primjer dizajniranja testnih slučajeva odabran je zahtjev koji govori da se u polje za unos bilješki tijekom izrade unosa glavobolje može unijeti maksimalno 1000 znakova. Kako bi se zahtjev potpuno testirao, potrebno je dizajnirati testne slučajeve u kojima će se provjeriti broj znakova u polju za unos bilješke nakon unosa manje od 1000 znakova, točno 1000 znakova i više od 1000 znakova.

Dizajniranje testnih slučajeva započinje određivanjem preduvjeta koji moraju biti zadovoljeni kako bi se testovi mogli izvoditi. Nakon toga, definirane se sve akcije koje se trebaju izvesti u testovima te su, na kraju, definirani očekivani rezultati. U tablici 5.1. moguće je vidjeti definirane preduvjete, akcije i očekivane rezultate dizajniranih testnih slučajeva.

Tablica 5.1. *Definirani preduvjeti, akcije i očekivani rezultati testnih slučajeva dizajniranih za testiranje zahtjeva o maksimalnom broju dopuštenih znakova u polju za unos bilješke*

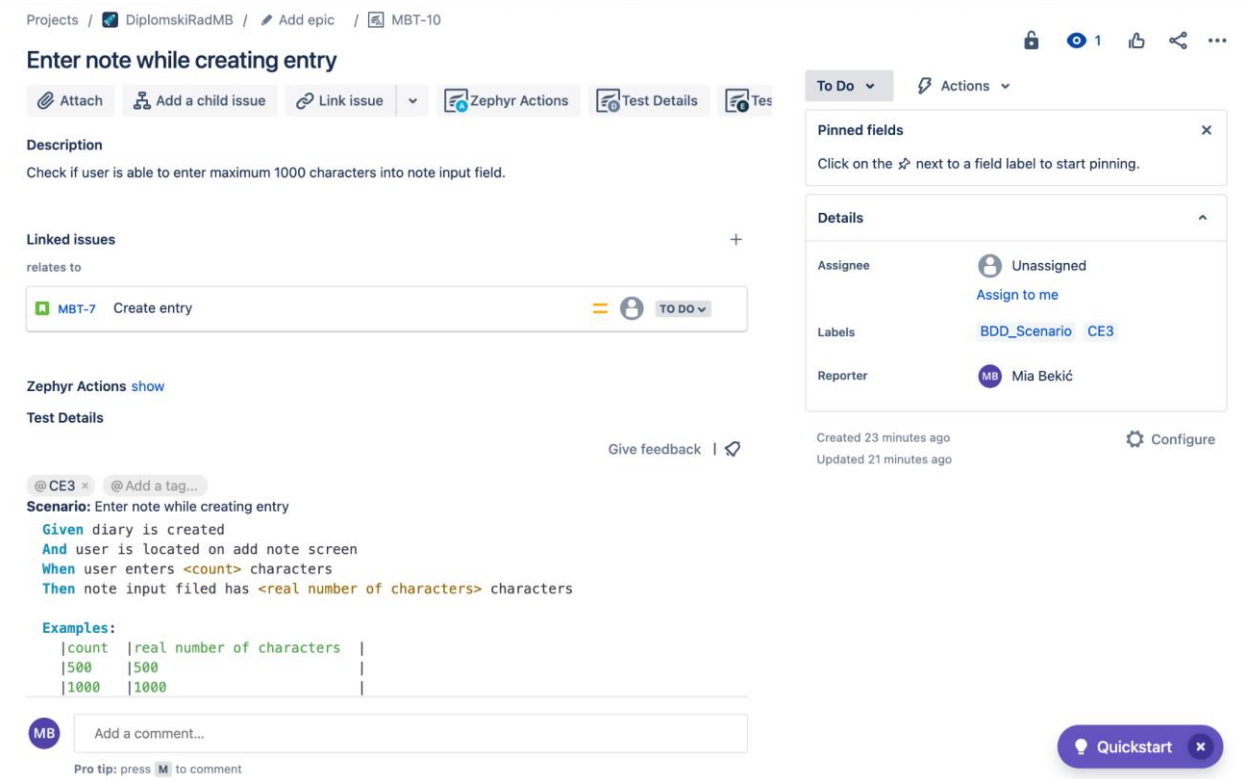
	Preduvjeti	**Akcije**	**Očekivani rezultati**
Testni slučaj 1	• dnevnik glavobolja mora biti izrađen • korisnik mora biti pozicioniran na zaslonu gdje može unijeti bilješku	• unos 500 znakova u polje za unos bilješke	• polje za unos bilješke sadrži 500 znakova
Testni slučaj 2	• dnevnik glavobolja mora biti izrađen • korisnik mora biti pozicioniran na zaslonu gdje može unijeti bilješku	• unos 1000 znakova u polje za unos bilješke	• polje za unos bilješke sadrži 1000 znakova
Testni slučaj 3	• dnevnik glavobolja mora biti izrađen • korisnik mora biti pozicioniran na zaslonu gdje može unijeti bilješku	• unos 1001 znakova u polje za unos bilješke	• polje za unos bilješke sadrži 1000 znakova

Zatim, pronađen je odgovarajući *story* problem u čiji sadržaj značajke je potrebno dodati definirane testne slučajeve. Nakon toga, definirane testne slučajeve bilo je potrebno prevesti u Gherkin sintaksu. S obzirom da se testni slučajevi, uzeti kao primjer, sastoje od istog obrasca koraka, samo s različitim ulaznim vrijednostima ili očekivanim ishodom, korišten je *Scenario Outline.* Nakon odabira naziva za *Scenario Outline*, definirani preduvjeti zapisani su u *Given* korake, akcije u *When* korake, a očekivani rezultati u *Then* korake. Također, *Scenario Outlineu* je dodana jedinstvena oznaka (engl. *tag*), u ovom primjeru *@CE3*. Spomenuta oznaka dodana je kako bi Zephyr Squad alat, prilikom uvoza rezultata, znao prepoznati kojem scenariju pripada koji rezultat. Izgled definiranog i dodanog *Scenario Outlinea*, u *story* problem, moguće je vidjeti na slici 5.2.



Slika 5.2. *Izgled dizajnirane testne skripte u Zephyr Squad alatu.*

Nakon dodavanja scenarija u sadržaj značajke *story* problema, automatski se izradio zasebni problem koji predstavlja dodani scenarij kojeg je zatim moguće dodavati u testne cikluse i izvoditi. Novi izrađeni problem prikazan je na slici 5.3.



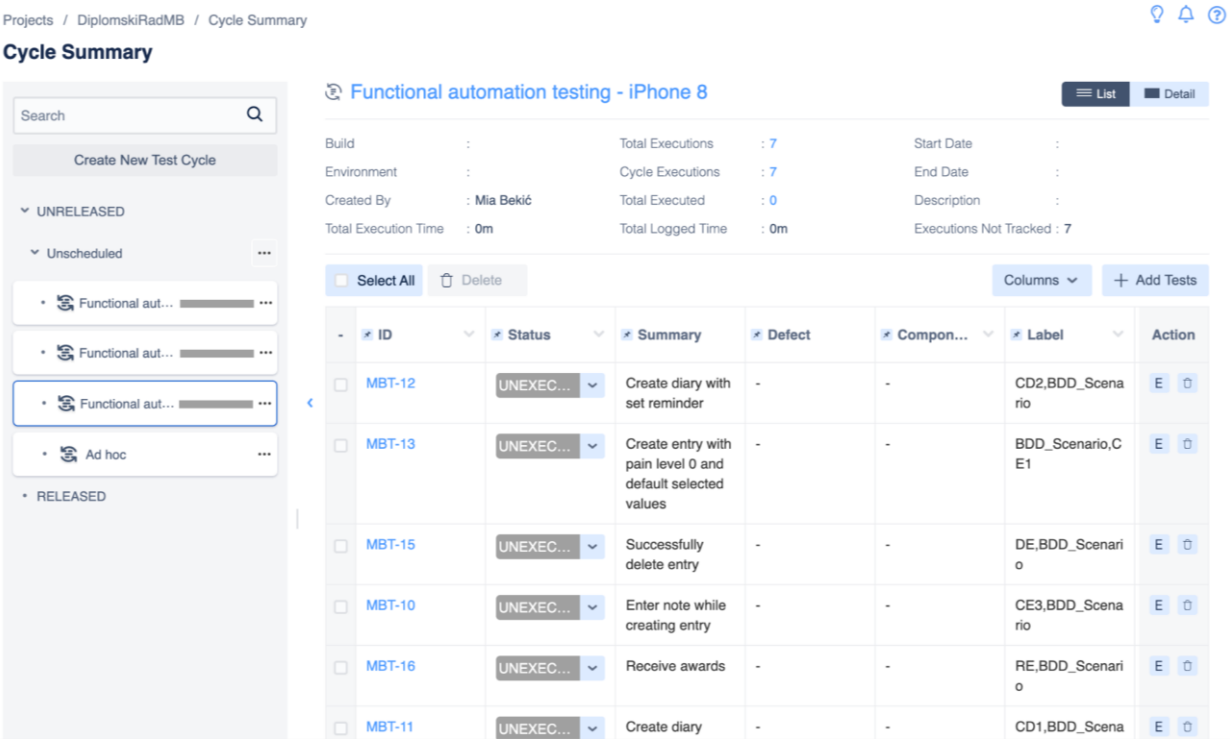
Slika 5.3. *Automatski izrađen problem nakon dodavanja scenarija u sadržaj značajke.*

Svi ostali testni slučajevi dizajnirani su na isti način, a ukupno je dizajnirano 13 testnih slučajeva navedenih u tablici 5.2.

Tablica 5.2. *Dizajnirani testni slučajevi za potrebe izvođenja automatiziranog testiranja.*

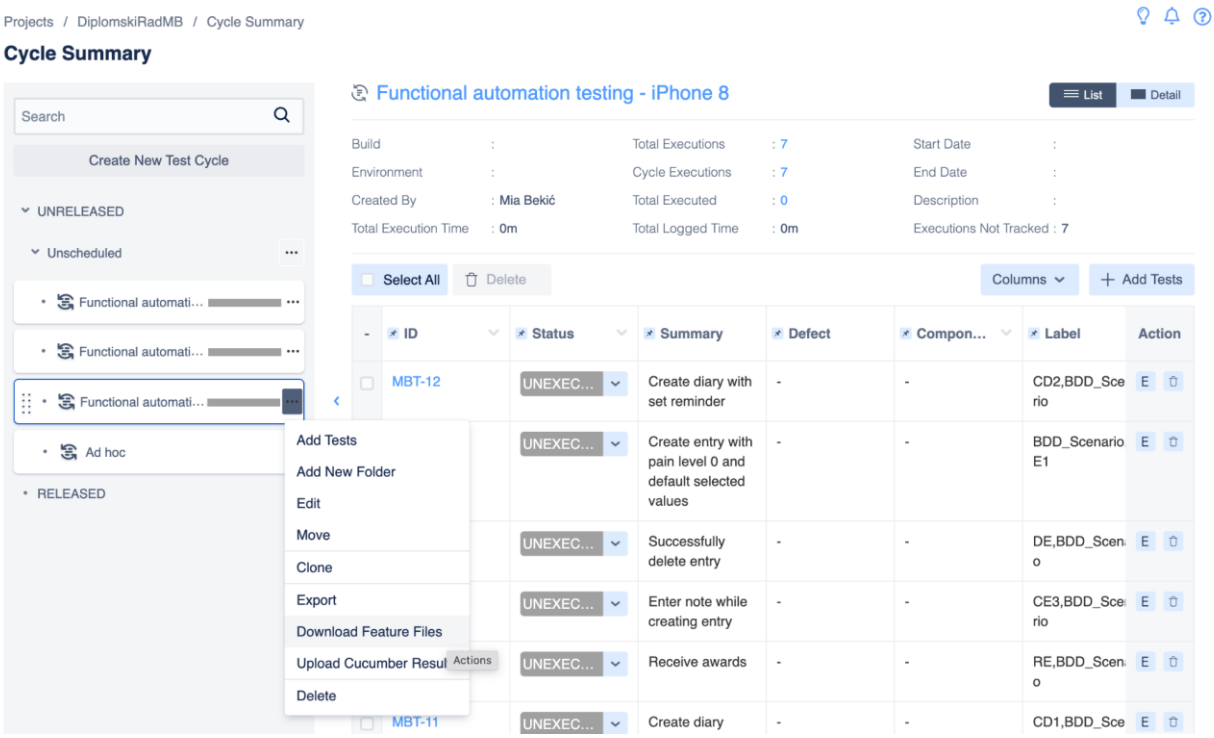
Što se testira?	Testni slučaj
Izrada dnevnika s postavljenim podsjetnikom	Uspješna izrada dnevnika s postavljenim podsjetnikom
Izrada dnevnika bez postavljenog podsjetnika	Uspješna izrada dnevnika bez postavljenog podsjetnika
Izrada unosa glavobolje s unesenim različitim podacima	Uspješna izrada unosa glavobolje s jačinom glavobolje 0 i ostalim predefiniranim vrijednostima
	Uspješna izrada unosa glavobolje s jačinom glavobolje 1, pozicijom glavobolje „Left side“, tipom boli „Pull“ i nuspojavom „Not specified“
	Uspješna izrada unosa glavobolje s jačinom glavobolje 2, pozicijom glavobolje „Bilateral“, tipom boli „Dull“ i nuspojavom „Dizziness“
Mogućnost unosa bilješke duljine maksimalno 1000 znakova prilikom izrade unosa glavobolje	Unos bilješke s 500 znakova
	Unos bilješke s 1000 znakova
	Unos bilješke s 1001 znakova
Dobivanje nagrada	Dobivanje nagrade nakon izrade 1 unosa o glavobolji s postavljenom vrijednosti razloga pojave glavobolje
	Dobivanje nagrade nakon izrade 10 unosa o glavobolji s postavljenom vrijednosti razloga pojave glavobolje
	Dobivanje nagrade nakon izrade 30 unosa o glavobolji s postavljenom vrijednosti razloga pojave glavobolje
Brisanje unosa glavobolje	Uspješno brisanje unosa o glavobolji s jačinom glavobolje 1 i pozicijom boli „Left side“
	Uspješno brisanje unosa o glavobolji s jačinom glavobolje 3 i pozicijom boli „Right side“

Od 13 testnih slučajeva, u Zephyr Squad alatu, izrađeno je 3 *Scenarija* i 4 *Scenario Outlinea*. Nakon dizajniranja testnih slučajeva, njihovog prevođenja u Gherkin sintaksu i unosa u Zephyr Squad alat, izrađeni su testni ciklusi u koje su dodani izrađeni scenariji (Slika 5.4.).



Slika 5.4. *Izrađeni testni ciklusi u Zephyr Squad alatu.*

Naposljetku, izrađeni scenariji preuzeti su iz Zephyr Squad alata, odabirom *Download Feature Files* opcije, s ciljem implementacije istih (Slika 5.5).



Slika 5.5. *Preuzimanje izrađenih scenarija iz Zephyr Squad alata*.

5.3. Implementacija automatiziranih testnih slučaja i izvođenje testiranja

Nakon dizajniranja testnih slučajeva i njihovog preuzimanja iz Zephyr Squad alata, izrađen je novi Java projekt pomoću IntelliJ IDE alata. S obzirom na odabrane tehnologije za korištenje, kao na primjer Appium, Cucumber i slično, u projekt su dodane potrebne ovisnosti (engl. *dependencies*). Za potrebe pokretanja lokalnog Appium poslužitelja, izrađena je klasa *ServiceManager* u kojoj su implementirane metode za pokretanje, zaustavljanje i dohvaćanje poslužitelja. Na slici 5.6. prikazana je metoda za pokretanje poslužitelja.

```
public void startService() {
    AppiumServiceBuilder builder = new AppiumServiceBuilder();
    builder.usingAnyFreePort()
            .usingDriverExecutable(new File( pathname: "/usr/local/bin/node"))
            .withAppiumJS(new File( pathname: "/usr/local/lib/node_modules/appium"));
    service.set(AppiumDriverLocalService.buildService(builder));
    service.get().start();
}
```

Slika 5.6. *Metoda startService za pokretanje lokalnog Appium poslužitelja.*

Željene mogućnosti sesija spremljene su u *.properties* dokumente, a sadržaj jednog od njih prikazan je na slici 5.7. Prilikom izrade drivera, prikazani podaci se dohvaćaju i predaju metodi za postavljanje drivera.

```
platformName=iOS
platformVersion=16.1
deviceName=iPhone 8
app=...
automationName= XCUITest
UDID=9F44A358-1121-412B-B662-B09BC0D163C9
bundleID=...
autoGrantPermissions=true
autoAcceptAlerts=true
```

Slika 5.7. *Željene mogućnosti sesije.*

Kako je za izvođenje automatiziranih testova potrebno izvoditi interakcije s različitim elementima na zaslonima aplikacije, bilo je potrebno izraditi klase koje predstavljaju zaslone aplikacije i u njima pristupiti elementima zaslona te implementirati potrebne metode za interakciju s njima. Za identifikaciju i pristup elementima zaslona aplikacije korištena je Appium Inspector aplikacija opisana u poglavlju 3. Elemente je moguće pretraživati po *accessibility id*, *iOS class chain* i drugim selektorima ili atributima. Inicijalizacija elemenata odrađena je korištenjem *@iOSXCUITFindBy* anotacije koja služi za pronalazak elemenata (Slika 5.8.)

```
@iOSXCUITFindBy(accessibility = "Let's go")
private WebElement createHeadacheDiaryBtn;
```

Slika 5.8. *Inicijalizacija elementa korištenjem @iOSXCUITFindBy anotacije.*

Sve zajedničke metode klasa koje predstavljaju zaslone aplikacije, izdvojene su u osnovnu klasu *BasePage* koju nasljeđuju sve ostale klase koje predstavljaju zaslone aplikacije. Neke od metoda koje *BasePage* implementira su: *click(WebElement), waitOnElementVisibility(WebElement), scrollUntilElementFoundOrEndOfPage(WebElement), isDisplayed(WebElement)* i druge.

- *click(WebElement)* – koristi se za pritiskanje elementa uz prethodno čekanje na vidljivost tog elementa.
- *waitOnElementVisibility(WebElement)* – koristi se za čekanje na vidljivost predanog elementa kao parametar.

- *scrollUntilElementFoundOrEndOfPage(WebElement)* – koristi se za listanje po zaslonu aplikacije dokle god predani element, kao parametar, nije vidljiv ili dok se ne dođe do kraja zaslona.
- *isDisplayed(WebElement)* – koristi se za dobivanje informacije o tome je li predani element, kao parametar, vidljiv ili ne.

Na slici 5.9. prikazane su metode *scrollUntilElementFoundOrEndOfPage(WebElement)* i *click(WebElement)*. Za potrebe ovog diplomskog rada izrađeno je 11 klasa koje predstavljaju zaslone aplikacije, a u njima su inicijalizirani samo oni elementi koji se koriste prilikom testiranja.

```
public void click(WebElement element){
    waitOnElementVisibility(element);
    element.click();
}

13 usages
public void scrollUntilElementFoundOrEndOfPage(WebElement element) {
    String previousPageSource = "";
    while (!isDisplayed(element) && !isEndOfPage(previousPageSource)) {
        previousPageSource = DriverManager.getDriver().getPageSource();
        scroll();
    }
}
```

Slika 5.9. *Metode click(WebElement element) i scrollUntilElementFoundOrEndOfPage(WebElement element).*

Nakon izrade klasa koje predstavljaju zaslone aplikacije, preuzeti scenariji iz Zephyr Squad alata dodani su u izrađeni projekt. Na slici 5.10. moguće je vidjeti prikaz jednog *.feature* dokumenta dodanog u projekt.

```
Feature: Create diary

  @CD1
  Scenario: Create diary without set reminder
    Given user is located on create diary screen
    And user data are prefilled
    When user clicks on continue button
    Then user is redirected to diary overview screen
    And reminder for entering diary entries is not set

  @CD2
  Scenario: Create diary with set reminder
      Given user is located on create diary screen
      And user data are prefilled
      When user clicks on switch button for setting reminders
      When user gives permission for sending notifications
      When user clicks on continue button
      Then user is redirected to diary overview screen
      And reminder for entering diary entries is set
```

Slika 5.10. *Prikaz createDiary.feature dokumenta.*

Zatim, izrađena je mapa *steps* u kojoj su izrađene dvije klase: *Hooks* i *StepDefinitions*. U *Hooks* klasi, implementirane su metode koje se izvode prije i poslije svakog scenarija, a navedeno je omogućeno korištenjem *@Before* i *@After* anotacija. U metodi *initialize()*, koja se izvodi na početku svakog scenarija, pokreće se aplikacija, dok se u metodi *quit()*, koja se izvodi na kraju svakog scenarija, aplikacija zatvara. Cilj *Hooks* klase je spriječiti dupliciranje koda. Klasa *StepDefinitions* predstavlja klasu u kojoj su implementirani koraci scenarija zapisanih u *.feature* dokumentima. Svaki korak *.feature* dokumenta mora imati odgovarajuću metodu u *StepsDefinitions* klasi jer, prema [34], Cucumber prilikom izvođenja koraka traži podudarajuću definiciju koraka za izvođenje. Svaka metoda mora imati Gherkin izraz (engl. *expression*) koji ga, prema [34], povezuje s jednim ili više Gherkin koraka. Gherkin izraz započinje anotacijom koja je jednaka korištenoj ključnoj riječi u koraku koji se nastoji implementirati. Zatim se u zagradi pod dvostrukim navodnicima navodi ostatak teksta koji predstavlja dio koraka. Nakon navedenog Gherkin izraza slijedi implementacija metode. U metodama se izvode razne provjere i interakcije s elementima aplikacije. Na slici 5.11. prikazane su *userClicksOnContinueButton()* i *userIsRedirectedToDiaryOverviewScreen()* metode s pridodanim Gherkin izrazom. Metoda *userClicksOnContinueButton()* s pridodanim Gherkin izrazom predstavlja definiciju koraka "*When user clicks on continue button*", a metoda izvodi pritisak na potvrdni gumb za izradu dnevnika na zaslonu za izradu dnevnika. Metoda *userIsRedirectedToDiaryOverviewScreen()* s pridodanim Gherkin izrazom predstavlja definiciju koraka "*Then user is redirected to diary*

overview screen", a metoda izvodi provjeru je li korisnik stvarno premješten na zaslon s naslovom "*Headache*".

```
3 usages
@When("user clicks on continue button")
public void userClicksOnContinueButton() {
    createDiaryPage.clickOnCreateDiaryBtn();
}
5 usages
@Then("user is redirected to diary overview screen")
public void userIsRedirectedToDiaryOverviewScreen() {
    Assert.assertEquals(overviewPage.getOverviewTitle()
            .getAttribute( name: "label"),  expected: "Headache");
}
```

Slika 5.11. *Primjer izrađenih definicija koraka.*

Nakon definiranja i izrade prethodno spomenutih zaslon klasa, mape *steps* te njenog sadržaja, izrađena je mapa *runners* u kojoj su izrađene klase odgovorne za izvođenje *.feature* dokumenata i koordiniranje koraka definiranih u tim datotekama s odgovarajućim definicijama koraka. Kako je već ranije navedeno, za testiranje su korištena 3 simulatora pa je tako bilo potrebno izraditi i tri klase za svakog od njih. Izrađene su klase *iPhone8TestRunner*, *iPhone11TestRunner* i *iPhone12ProTestRunner* te *BaseRunner* klasa. *BaseRunner* klasa implementira dvije metode od kojih *setUp(String, Boolean)* metoda služi za pokretanje Appium poslužitelja i postavljanja drivera prije izvođenja testova, a *tearDown()* metoda služi za zaustavljanje Appium poslužitelja i odustajanje od drivera nakon izvođenja testova. Za izvođenje metoda prije i poslije izvođenja testova korištene su TestNG anotacije *@BeforeClass* i *@AfterClass*. Osim spomenutih anotacija, korištena je i *@Parametar* anotacija za predavanje parametara metodi *setUp(String, Boolean)* kroz *testng.xml* dokument. Na slici 5.12. prikazana je metoda *setUp(String, Boolean)*. Prvi parametar koji se predaje prikazanoj metodi predstavlja naziv *.properties* dokumenta u kojem se nalaze definirane željene mogućnosti sesije. Drugi parametar koji se predaje metodi predstavlja *Boolean* vrijednost koja označava koristi li se udaljeni Appium poslužitelj ili ne. Ukoliko se ne koristi udaljeni poslužitelj, u metodi se pokreće lokalni poslužitelj i njegova URL adresa se koristi prilikom postavljanja drivera. Ukoliko se koristi udaljeni poslužitelj, u *remote_server.properties* dokumentu mora biti definirana URL adresa udaljenog poslužitelja koja se iz njega dohvaća i koristi za postavljanje drivera.

```
@Parameters({"configFile", "remoteServer"})
@BeforeClass
public void setUp(String configFile, Boolean remoteServer) throws Exception {
    capabilities = new DesiredCapabilitiesManager();
    if(!remoteServer){
        properties = new PropertiesManager(configFile);
        capabilities = new DesiredCapabilitiesManager();
        capabilities.setCapabilities(properties);
        server = new ServiceManager();
        server.startService();
        DriverManager
                .setDriver(server.getServerURL(),
                        capabilities.getDesiredCapabilities());
    } else {
        properties = new PropertiesManager(configFile, remoteServerURL: "remote_server.properties");
        capabilities.setCapabilities(properties);
        DriverManager
                .setDriver(new URL(properties.getAppiumURL()),
                        capabilities.getDesiredCapabilities());
    }

}
```

Slika 5.12. *Prikaz implementacije setUp(String, Boolean) metode.*

Sve ostale klase iz *runners* mape nasljeđuju *BaseRunner* klasu i u njima se definiraju željene konfiguracije runnera pomoću *@CucumberOptions* anotacije. Prilikom definiranja konfiguracija zahtjeva se definiranje opcija *features* i *glue*. Prilikom definiranja opcije *features* potrebno je predati putanju do *.feature* dokumenata dok je prilikom definiranja *glue* opcije potrebno predati paket (engl. *package*) gdje se nalaze definicije koraka i *Hooks* klasa. Cucumber također daje opciju korištenja dodataka za izvještavanje, a njihovo korištenje definira se u *plugin* opciji. Za izvještavanje, od ugrađenih opcija dodataka koji se mogu koristiti, odabrani su *pretty*, *html* i *json* dodatci za formatiranje. Također, za detaljnije izvještavanje koje prikazuje razne dijagrame, nudi podatke o vremenu izvođenja pojedine značajke, scenarija, koraka i slično, korišten je dodatak dostupan na https://github.com/damianszczepanik/cucumber-reporting. Na slici 5.13. moguće je vidjeti *iPhone8TestRunner* klasu i definiranu konfiguraciju tog runnera.

```
@CucumberOptions(
        plugin = {
                "pretty",
                "json:target/cucumber-reports/iPhone8-report.json",
                "html:target/cucumber-reports/iPhone8-report.html",
                "me.jvt.cucumber.report.PrettyReports:target/cucumber-reports/cucumber-html-report-iPhone8",
        },
        features = {"src/test/resources/features"},
        glue = {"steps"},
        monochrome = true
)
public class iPhone8TestRunner extends BaseRunner { }
```

Slika 5.13. *iPhone8TestRunner klasa.*

Prije izvođenja testiranja, bilo je potrebno konfigurirati *testng.xml* dokument. Navedeni dokument, prema [35], pomaže u organizaciji testova te omogućava izradu i rukovanje testnim klasama, definiranje testnih paketa (engl. *test suites*) i testova. Konfigurirani *testng.xml* dokument korišten za izvođenje automatiziranog testiranja korištenjem lokalnog Appium poslužitelja moguće je vidjeti na slici 5.14.

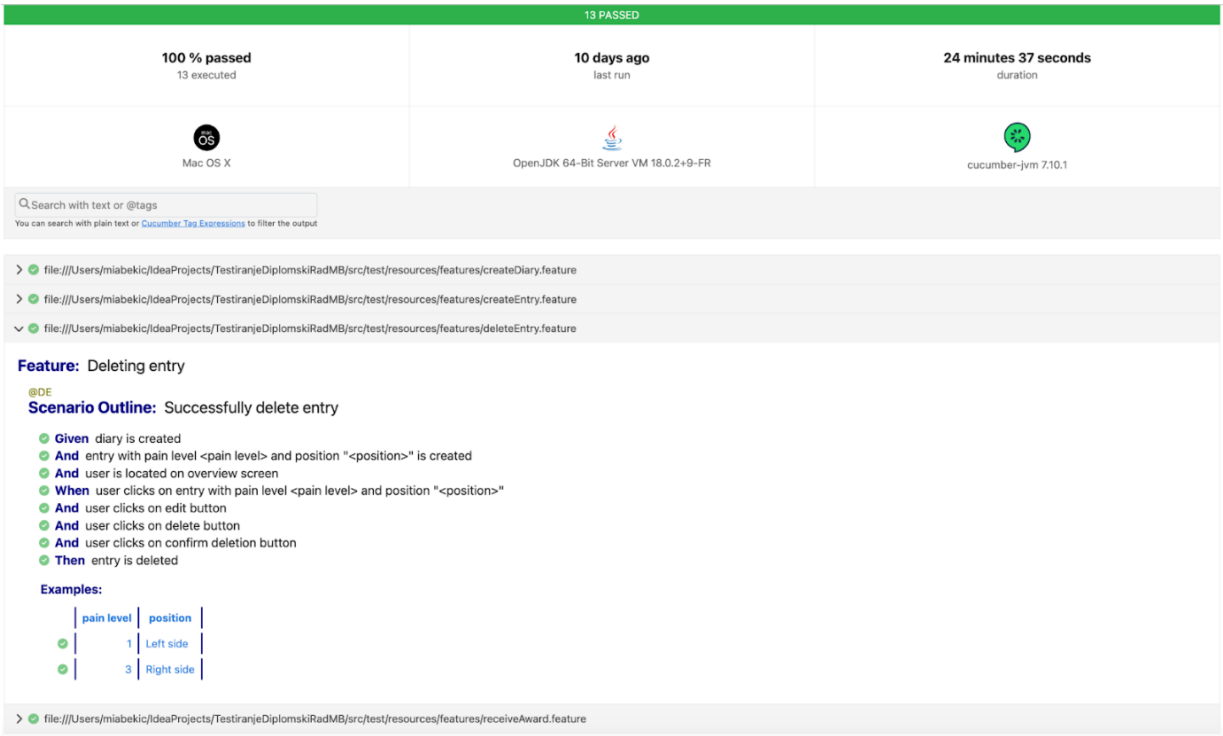
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Functional testing">
    <test name="iPhone 8">
        <parameter name="configFile" value="iPhone_8_simulator_desired_cap.properties" />
        <parameter name="remoteServer" value="false"/>
        <classes>
            <class name="runners.iPhone8TestRunner"/>
        </classes>
    </test>
    <test name="iPhone 11">
        <parameter name="configFile" value="iPhone_11_simulator_desired_cap.properties" />
        <parameter name="remoteServer" value="false"/>
        <classes>
            <class name="runners.iPhone11TestRunner"/>
        </classes>
    </test>
    <test name="iPhone 12 Pro">
        <parameter name="configFile" value="iPhone_12_Pro_simulator_desired_cap.properties" />
        <parameter name="remoteServer" value="false"/>
        <classes>
            <class name="runners.iPhone12ProTestRunner"/>
        </classes>
    </test>
</suite>
```

Slika 5.14. *testng.xml dokument.*

Nakon konfiguracije *testng.xml* dokumenta, unutar IntelliJ IDE isti je odabran za izvođenje te je pritisnut gumb za izvođenje čime je započelo izvođenje definiranih scenarija.

6. REZULTATI TESTIRANJA

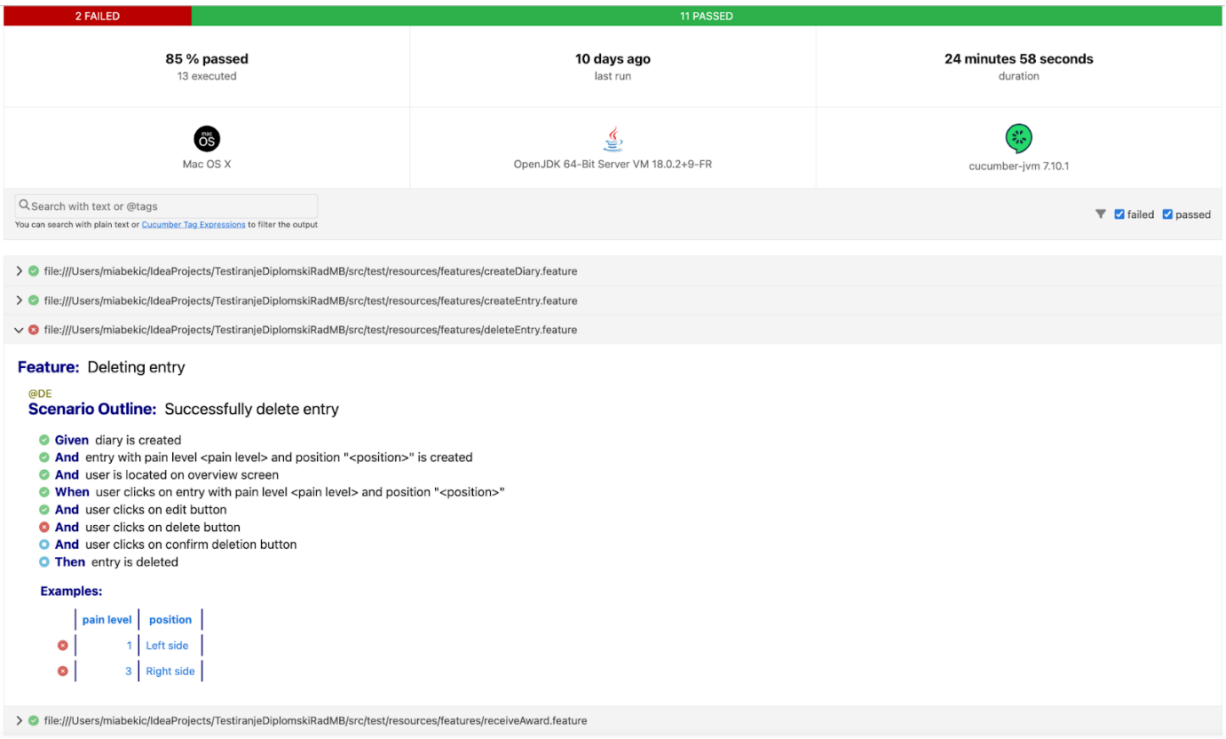
Prvo izvedeno automatizirano testiranje provedeno je korištenjem lokalnog Appium poslužitelja. Nakon što je testiranje završilo, dobiveni su rezultati testiranja. Zahvaljujući korištenim dodacima za formatiranje, generirani su izvještaji iz kojih je moguće pregledati rezultate testiranja. Na slici 6.1. prikazan je izvještaj provedenog automatiziranog testiranja na simulatoru iPhone 8 (iOS 16.1) korištenjem lokalnog Appium poslužitelja. Izvještaj omogućava uvid u: postotak prolaznosti testova, vrijeme zadnjeg testiranja, trajanje cjelokupnog testiranja, operacijski sustav uređaja pomoću kojeg se pokrenulo testiranje, sadržaj svake izvedene značajke, status prolaska svake značajke, scenarija, koraka i slično. Također, značajke i njihov sadržaj moguće je pretraživati korištenjem teksta ili oznaka. Iz izvještaja vidljiva je prolaznost testova od 100% što znači da su svi izvedeni testovi prošli.



Slika 6.1. *Izvješće automatiziranog testiranja provedenog na simulatoru iPhone 8 (iOS 16.1) korištenjem lokalnog Appium poslužitelja.*

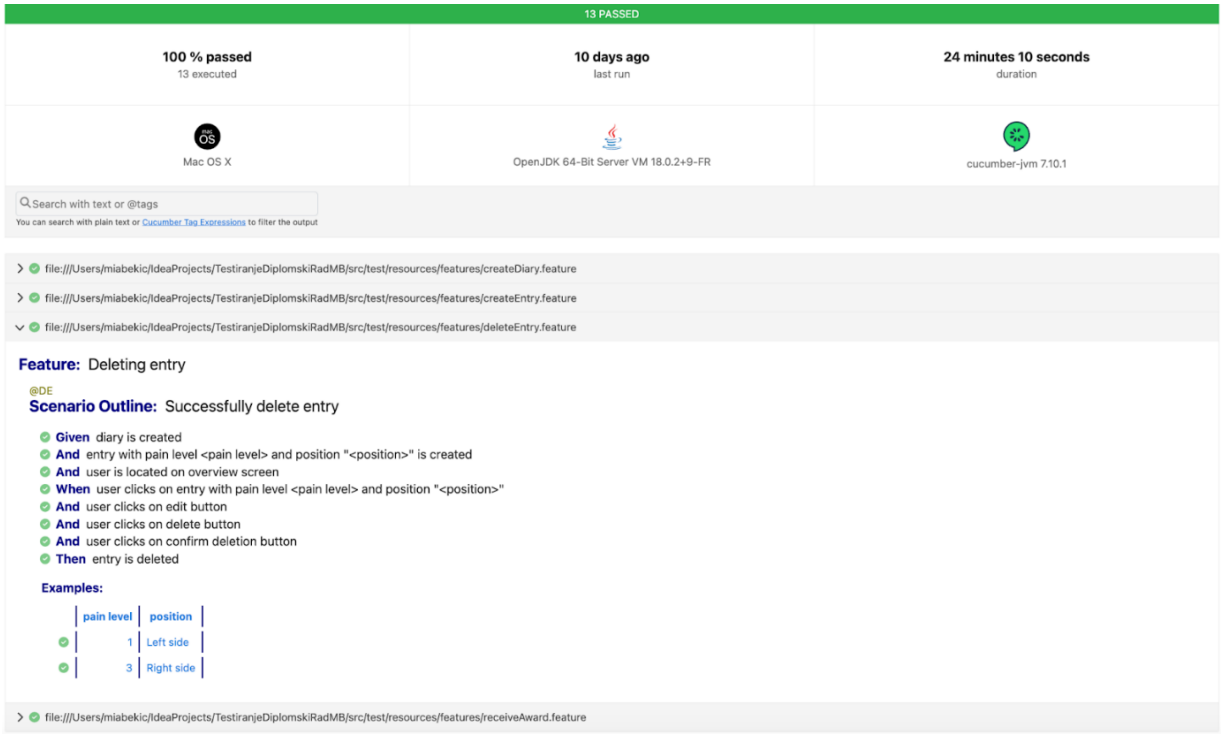
Na slici 6.2. prikazan je izvještaj izvedenog automatiziranog testiranja na simulatoru iPhone 12 Pro (iOS 16.4) korištenjem lokalnog Appium poslužitelja. Iz izvještaja moguće je vidjeti kako je značajka *Deleting entry* pala prilikom izvođenja oba scenarija prilikom pokušaja izvođenja koraka "*And user clicks on delete button*". Sve ostale izvedene značajke su prošle što daje prolaznost testova od 85%. Zbog postojanja oba statusa prolaznosti u izvještaju, izvještaj daje mogućnost

filtriranja značajki po statusu prolaznosti označavanjem odgovarajuće potvrdnog okvira (engl. *checkbox*).



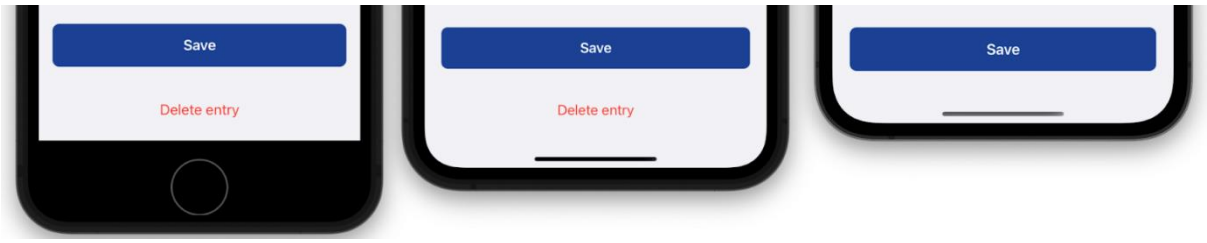
Slika 6.2. *Izvješće automatiziranog testiranja provedenog na simulatoru iPhone 12 Pro (iOS 16.4) korištenjem lokalnog Appium poslužitelja.*

Na slici 6.3. prikazan je izvještaj izvedenog automatiziranog testiranja na simulatoru iPhone 11 (iOS 16.2) korištenjem lokalnog Appium poslužitelja. Iz izvještaja moguće je vidjeti prolaznost testova od 100%.



Slika 6.3. *Izvješće automatiziranog testiranja provedenog na simulatoru iPhone 11 (iOS 16.2) korištenjem lokalnog Appium poslužitelja.*

Na temelju analize rezultata, može se zaključiti kako postoji jedan defekt kojeg je potrebno prijaviti razvojnom timu. S obzirom da se defekt pojavio prilikom testiranja na simulatoru s iOS verzijom 16.4, a prilikom testiranja na simulatorima s iOS verzijama 16.1 i 16.2 ne, zaključak je kako je defekt povezan s verzijom operacijskog sustava. Do pojave defekta došlo je zbog nepostojanja gumba za brisanje unosa glavobolje (*Delete entry* gumb), slika 6.4.



Slika 6.4. *Nepostojanje gumba za brisanje unosa glavobolje.*

Analizom potrebnog vremena za izvođenje pojedinih značajki uočeno je kako se značajka ***Receive award*** izvodila najduže (Slika 6.5.). Navedeno je bilo i očekivano jer za osvajanje određene nagrade treba izraditi određeni broj unosa što zahtjeva dosta vremena. Najkraće potrebno vrijeme za izvođenje *Receive award* značajke ostvareno je na simulatoru iPhone 8 (15 minuta 2 sekunde i 462 milisekunde), dok je najduže potrebno vrijeme za izvođenje navedene značajke ostvareno na

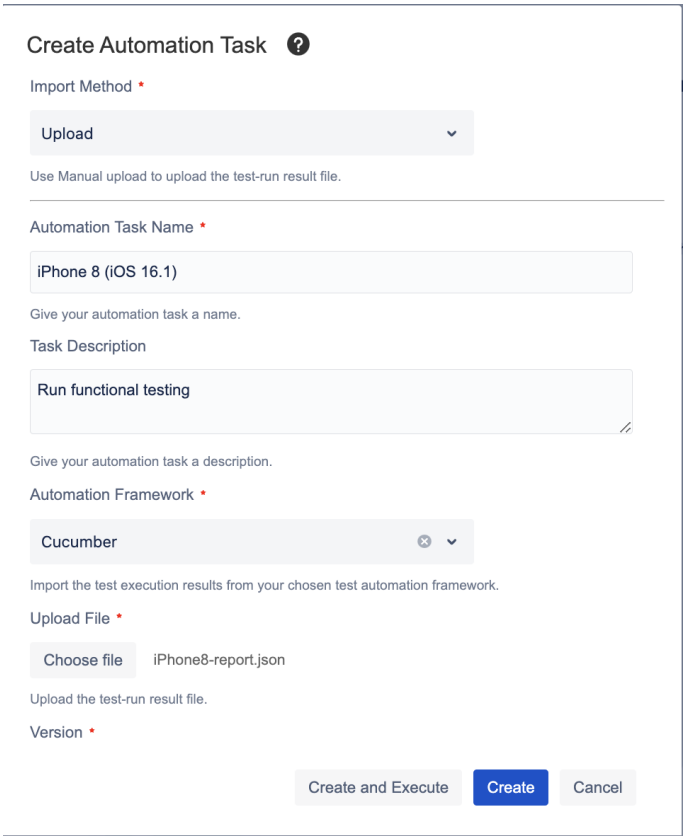
simulatoru iPhone 12 Pro (16 minuta 55 sekundi i 629 milisekundi). Značajka koja se najkraće izvodila jest ***Create diary*** što je također bilo očekivano jer je za izradu dnevnika potrebno samo par pritisaka na par gumbova i dnevnik je izrađen. Najkraće potrebno vrijeme za izvođenje *Create diary* značajke ostvareno je na simulatoru iPhone 12 Pro (15 sekundi i 943 milisekunde), dok je najduže potrebno vrijeme za izvođenje navedene značajke ostvareno na simulatoru iPhone 8 (24 sekunde i 21 milisekunde). Najdulje sveukupno potrebno vrijeme za izvođenje svih značajki ostvareno je prilikom izvođenja na simulatoru **iPhone 12 Pro**, dok je najkraće sveukupno potrebno vrijeme za izvođenje svih značajki ostvareno prilikom izvođenja na simulatoru **iPhone 8**. Različita vremena potrebna za izvođenje testiranja značajki ponajviše ovise o potrebnom vremenu za pronalaženje određenog elementa na zaslonu.

	iPhone 8 (iOS 16.1)		iPhone 11 (iOS 16.2)		iPhone 12 Pro (iOS 16.4)	
Značajka	**Trajanje**	**Status**	**Trajanje**	**Status**	**Trajanje**	**Status**
Create diary	24.021	Prošao	17.205	Prošao	15.943	Prošao
Create entry	3:44.038	Prošao	2:12.605	Prošao	2:48.702	Prošao
Deleting entry	1:32.376	Prošao	1:42.770	Prošao	2:2.786	Pao
Receive award	15:2.462	Prošao	16:55.323	Prošao	16:55.629	Prošao
	20:42.897	**4**	**21:7.905**	**4**	**22:3.061**	**4**
		100.00%		**100.00%**		**75.00%**

Slika 6.5. *Vremena potrebna za izvođenje pojedinih i svih značajki prilikom izvođenja na odabranim simulatorima.*

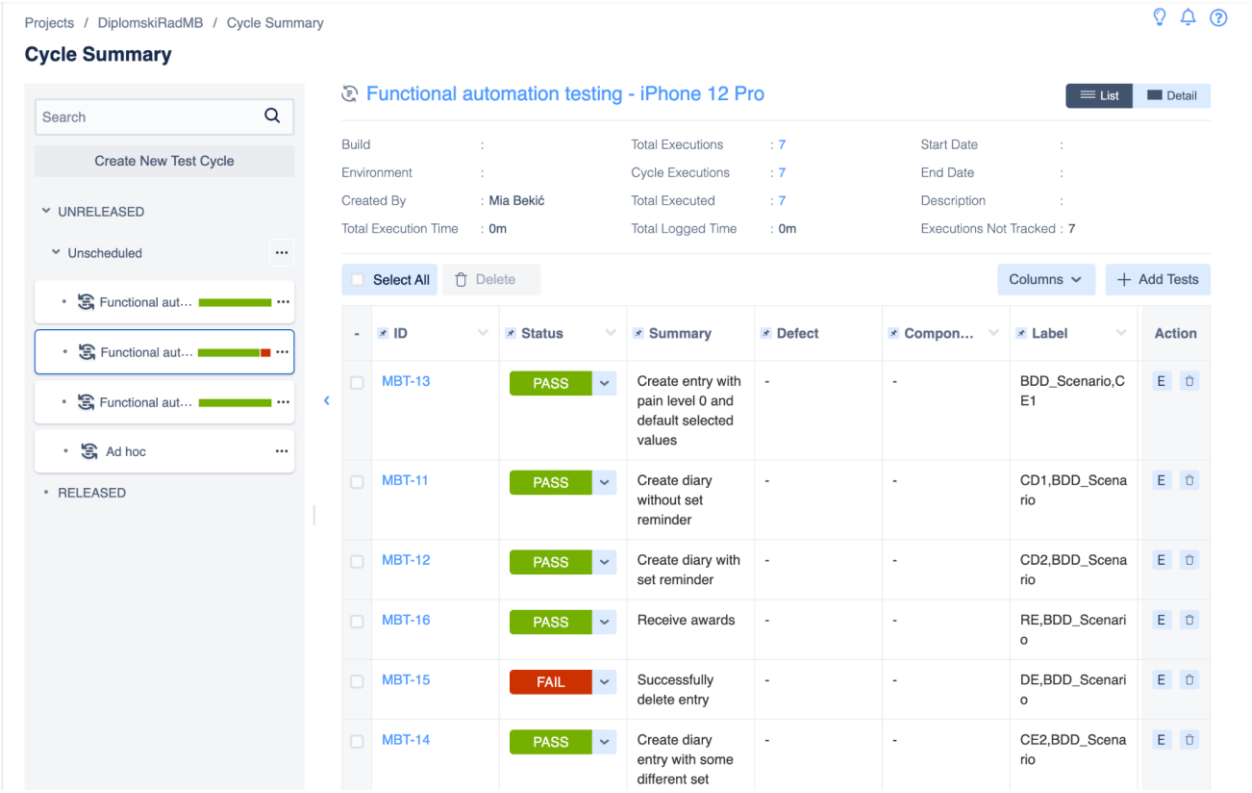
6.1. Uvoz rezultata automatiziranog testiranja u Zephyr Squad alat

Kako bi se rezultati provedenog automatiziranog testiranja uvezli u Zephyr Squad alat, potrebno je izraditi zadatak za automatizaciju (engl. *automation task*). Prilikom izrade zadatka za automatizaciju, prvo je potrebno definirati uvoznu metodu. Postoje dvije uvozne metode, a to su: *upload* i *Zbot*. [27] Na temelju odabrane uvozne metode određuje se što je još potrebno definirati kako bi se izradio zadatak za automatizaciju. U sklopu ovog diplomskog rada, za uvoznu metodu odabrana je opcija *upload* koja zahtjeva definiranje: naziva zadatka za automatizaciju, testnog okvira iz kojeg se uvoze rezultati, dokumenta koji se uvozi, verzije u koju se uvoze rezultati, testnog ciklusa u koji se uvoze rezultati te reportera. Ostale vrijednosti koje se nude nije nužno definirati. Na slici 6.6. prikazan je djelomični izgled prozora u kojem je moguće izraditi zadatak za automatizaciju. Nakon unosa svih potrebnih vrijednosti, pritisnut je gumb *Create and Execute* kojim se izradio zadatak za automatizaciju i kojim se odmah započelo s izvođenjem zadatka.



Slika 6.6. *Djelomični izgled prozor za izradu zadatka za automatizaciju.*

Izrađena su i izvedena 3 zadatka za automatizaciju. Nakon njihovog izvršenja, rezultati automatiziranog testiranja primijenjeni su na testnim slučajevima iz testnih ciklusa (Slika 6.7.). Kao i u prethodnim izvještajima, moguće je vidjeti kako svi testovi imaju status prolaza osim testova za brisanje unosa glavobolje izvedenih na simulatoru iPhone 12 Pro (iOS 16.4). Navedeni defekt potrebno je prijaviti razvojnom timu što je u Jiri vrlo jednostavno. Potrebno je samo izraditi novi problem tipa *bug* u kojem je potrebno definirati: što je zapravo problem, korake kako reproducirati defekt, očekivani rezultat, dobiveni rezultat, naziv i verziju aplikacije u kojoj je defekt pronađen, uređaj na kojem je testiranje izvedeno te priložiti slike ili snimke zaslona kao dokaz o postojanju defekta. Zahvaljujući Jiri i Zephyr Squad alatu, svi sudionici projekta mogu imati uvid u rezultate provedenih testiranja.



Slika 6.7. *Stanje testnih ciklusa nakon uvoza rezultata automatiziranog testiranja.*

6.2. Usporedba izvedenog automatiziranog testiranja korištenjem lokalnog i udaljenog poslužitelja

Kako bi se koristio udaljeni Appium poslužitelj, prvo je potrebno u *testng.xml* dokumentu postaviti vrijednost parametra *remoteServer* na *true*. Zatim, potrebno je pokrenuti udaljeni Appium poslužitelj i odrediti IP adresu uređaja na kojem je poslužitelj pokrenut te zapisati URL, adresu preko koje je moguće pristupiti pokrenutom poslužitelju, u *remote_server.properties* dokument. Udaljeni Appium poslužitelj pokrenut je na prijenosnom računalu MacBook Pro, računalu koje ima pristup simulatorima na kojima se provodi testiranje, a skripta se izvodila na prijenosnom računalu Asus VivoBook. Nakon što je testiranje završeno, generirani su izvještaji te su dobiveni rezultati uspoređeni s rezultatima dobivenim prilikom izvođenja testiranja korištenjem lokalnog Appium poslužitelja. S obzirom da se samo promijenio korišteni Appium poslužitelj, svi testovi imaju jednake statuse prolaznosti kao i nakon izvedenog testiranja korištenjem lokalnog Appium poslužitelja.

Na slici 6.8. prikazana je usporedba vremena potrebnog za izvođenje testiranja značajki na simulatoru iPhone 8 (iOS 16.1) prilikom korištenja lokalnog i udaljenog Appium poslužitelja. Iz

prikazanih podataka vidljivo je kako je prilikom korištenja udaljenog Appium poslužitelja, u usporedbi s lokalnim Appium poslužiteljem, došlo do povećanja potrebnog vremena za izvođenje testiranja svake značajke. Potrebna vremena za izvođenje testiranja prilikom korištenja udaljenog poslužitelja duža su za 3 sekunde i 406 milisekundi, 5 sekundi i 686 milisekundi, 43 sekunde i 61 milisekundi, 1 minutu 44 sekunde i 565 milisekundi, za značajke *Create diary*, *Create entry*, *Deleting entry* i *Receive award*. Kada se sve sagleda, vrijeme potrebno za izvođenje testiranja svih značajki korištenjem udaljenog poslužitelja duže je za **2 minute 36 sekundi i 720 milisekundi**.

	Lokalni poslužitelj		Udaljeni poslužitelj	
Značajka	**Trajanje**	**Status**	**Trajanje**	**Status**
Create diary	24.021	**Prošao**	27.427	**Prošao**
Create entry	3:44.038	**Prošao**	3:49.724	**Prošao**
Deleting entry	1:32.376	**Prošao**	2:15.437	**Prošao**
Receive award	15:2.462	**Prošao**	16:47.027	**Prošao**
	20:42.897	**4**	**23:19.617**	**4**
		100.00%		**100.00%**

Slika 6.8. *Usporedba potrebnog vremena za izvođenje testiranja pojedinih značajki na simulatoru iPhone 8 (iOS 16.1) prilikom korištenja lokalnog i udaljenog poslužitelja.*

Na slici 6.9. prikazana je usporedba vremena potrebnog za izvođenje testiranja značajki na simulatoru iPhone 11 (iOS 16.2) prilikom korištenja lokalnog i udaljenog Appium poslužitelja. Iz prikazanih podataka vidljivo je kako je prilikom korištenja udaljenog Appium poslužitelja, u usporedbi s lokalnim Appium poslužiteljem, došlo do povećava vremena potrebnog za izvođenje testiranja svake značajke. Potrebna vremena za izvođenje testiranja prilikom korištenja udaljenog poslužitelja duža su za 3 sekunde i 315 milisekundi, 17 sekundi i 83 milisekundi, 22 sekunde i 335 milisekundi, 1 minutu 36 sekunde i 65 milisekundi, za značajke *Create diary*, *Create entry*, *Deleting entry* i *Receive award*. Kada se sve sagleda, vrijeme potrebno za izvođenje testiranja svih značajki korištenjem udaljenog poslužitelja duže je za **2 minute 18 sekundi i 797 milisekundi**.

	Lokalni poslužitelj		Udaljeni poslužitelj	
Značajka	Trajanje	Status	Trajanje	Status
Create diary	17.205	Prošao	20.520	Prošao
Create entry	2:12.605	Prošao	2:29.688	Prošao
Deleting entry	1:42.770	Prošao	2:5.105	Prošao
Receive award	16:55.323	Prošao	18:31.388	Prošao
	21:7.905	**4**	**23:26.702**	**4**
		100.00%		**100.00%**

Slika 6.9. *Usporedba potrebnog vremena za izvođenje testiranja pojedinih značajki na simulatoru iPhone 11 (iOS 16.2) prilikom korištenja lokalnog i udaljenog poslužitelja.*

Na slici 6.10. prikazana je usporedba vremena potrebnog za izvođenje testiranja značajki na simulatoru iPhone 12 Pro (iOS 16.4) prilikom korištenja lokalnog i udaljenog Appium poslužitelja. Iz prikazanih podataka vidljivo je kako je prilikom korištenja udaljenog Appium poslužitelja, u usporedbi s lokalnim Appium poslužiteljem, došlo do povećava potrebnog vremena za izvođenje testiranja svake značajke osim značajke *Create entry*. Potrebna vremena za izvođenje testiranja prilikom korištenja udaljenog poslužitelja duža su za 2 sekunde i 841 milisekundi, 37 sekundi i 273 milisekundi, 1 minutu 24 sekunde i 94 milisekunde, za značajke *Create diary*, *Deleting entry* i *Receive award*. Potrebno vrijeme za izvođenje testiranja značajke *Create entry* prilikom korištenja udaljenog poslužitelja kraće je za 28 sekundi i 576 milisekundi. Kada se sve sagleda, vrijeme potrebno za izvođenje testiranja svih značajki korištenjem udaljenog poslužitelja duže je za **1 minutu 35 sekundi i 633 milisekunde**.

	Lokalni poslužitelj		Udaljeni poslužitelj	
Značajka	Trajanje	Status	Trajanje	Status
Create diary	15.943	Prošao	18.784	Prošao
Create entry	2:48.702	Prošao	2:20.126	Prošao
Deleting entry	2:2.786	Pao	2:40.059	Pao
Receive award	16:55.629	Prošao	18:19.723	Prošao
	22:3.061	**4**	**23:38.694**	**4**
		75.00%		**75.00%**

Slika 6.10. *Usporedba potrebnog vremena za izvođenje testiranja pojedinih značajki na simulatoru iPhone 12 Pro (iOS 16.4) prilikom korištenja lokalnog i udaljenog poslužitelja.*

Na temelju prikazane usporedbe, moguće je zaključiti kako je najveće povećanje potrebnog vremena za izvođenja testiranja korištenjem udaljenog poslužitelja postignuto testiranjem na simulatoru **iPhone 8 (iOS 16.1)**, dok je najmanje povećanje ostvareno testiranjem na simulatoru **iPhone 12 Pro (iOS 16.4)**. Povećanja potrebnog vremena za izvođenje testiranja značajki korištenjem udaljenog poslužitelja u usporedbi s korištenjem lokalnog poslužitelja su bila i očekivano jer se komunikacija s lokalnim poslužiteljem ostvaruje brže nego s udaljenim poslužiteljem. Također, pojavio se i slučaj gdje je došlo do smanjenja potrebnog vremena za izvođenje testiranja određene značajke korištenjem udaljenog poslužitelja u usporedbi s korištenjem lokalnog poslužitelja. Na brzinu pronalaska određenog elementa i na brzinu izvođenja određenih akcija, prilikom izvođenja testiranja, nije moguće utjecati, što zapravo objašnjava povećanja/smanjenja potrebnog vremena za izvođenje testiranja značajki.

7. ZAKLJUČAK

Cilj ovog diplomskog rada je automatizirano testiranje odabrane mobilne aplikacije korištenjem Appium okruženja. Osim Appium okruženja, korišteni su i Cucumber te Zephyr alati.

Odabrana aplikacija za testiranje u sklopu ovog diplomskog rada je zdravstvena aplikacija za vođenje dnevnika glavobolja. Zdravstvena aplikacija, kao i svaka druga aplikacija, zahtjeva testiranje kako bi se osigurala njena kvaliteta i ispravan rad. Za kvalitetno izvođenje testiranja, potrebno je definirati testni proces. U praksi, testni proces zdravstvenih aplikacije općenito se sastoji od koraka: planiranje testiranja, analiza zahtjeva, dizajniranje testnih slučajeva, funkcionalno i nefunkcionalno testiranje, izvještavanje. Važno je naglasiti da testni proces nije uvijek isti nego se prilagođava shodno potrebama projekta. Definirani testni proces ovog diplomskog rada sastoji se od koraka: planiranja testiranja, analize zahtjeva, dizajniranja testnih slučajeva i njihove implementacije, funkcionalnog testiranja i izvještavanja.

Prilikom izvođenja planiranja testiranja u sklopu ovog diplomskog rada, donesena je odluka o izvođenju funkcionalnog testiranja zdravstvene aplikacije za vođenje dnevnika o glavoboljama korištenjem automatiziranog testiranja, definirani su potrebni resursi i rizici te je definirana testna okolina. Nakon analize zahtjeva postavljanih na aplikaciju donesena je odluka što će se sve automatizirano testirati, a to je: izrada dnevnika s postavljenim podsjetnikom, izrada dnevnika bez postavljenog podsjetnika, izrada unosa glavobolje s unesenim različitim podacima, mogućnost unosa bilješke duljine maksimalno 1000 znakova prilikom izrade unosa glavobolje, dobivanje nagrada i brisanje unosa glavobolje. Na temelju odabranih funkcionalnosti za testiranje, dizajnirano je 13 testnih slučajeva koji su potom prevedeni u Gherkin sintaksu i uneseni u Zephyr Squad alat. Nakon provedenog dizajniranja testnih slučajeva izrađen je projekt za automatizirano testiranje. U projektu su izrađene potrebne klase i dokumenti te su implementirane metode potrebne za izvođenje automatiziranog testiranja. Nakon izrade projekta izvedeno je automatizirano testiranje na simulatorima iPhone 8 (iOS 16.1), iPhone 11 (iOS 16.2) i iPhone 12 Pro (iOS 16.4) korištenjem lokalnog i udaljenog poslužitelja.

Izvedenim automatiziranim testiranjem dobiveni su izvještaji iz kojih su očitani i uspoređeni rezultati testiranja. Iz rezultata testiranja uočen je jedan defekt, nastao prilikom testiranja funkcionalnosti brisanja unosa glavobolje. Defekt se pojavljuje samo prilikom testiranja na simulatoru s iOS verzijom 16.4, a događa se zbog nepostojanja gumba za brisanje unosa. Dobiveni rezultati testiranja uvezeni su u Zephyr Squad alat s ciljem omogućavanja kasnijeg pregledavanja

podataka o provedenim testiranjima. Od testiranih funkcionalnosti, najdulje potrebno vrijeme za izvođenje automatiziranog testiranja zahtjeva funkcionalnost dobivanja nagrada, dok najkraće vrijeme zahtjeva funkcionalnost izrade dnevnika. Usporedbom potrebnog vremena za izvođenje automatiziranog testiranja korištenjem lokalnog i udaljenog poslužitelja, uočena je razlika u potrebnom vremenu za izvođenje testiranja. U svim slučajevima, osim prilikom testiranja funkcionalnosti izrade unosa glavobolje na simulatoru iPhone 12 Pro, došlo je do povećanja potrebnog vremena za izvođenje automatiziranog testiranja određene funkcionalnosti prilikom korištenja udaljenog poslužitelja. Prilikom testiranja funkcionalnosti izrade unosa glavobolje na simulatoru iPhone 12 Pro, došlo je do smanjenja potrebnog vremena za izvođenje automatiziranog testiranja prilikom korištenja udaljenog poslužitelja. Najmanje povećanje potrebnog vremena za izvođenje automatiziranog testiranja korištenjem udaljenog poslužitelja uočeno je testiranjem na simulatoru iPhone 12 Pro (1 minuta 35 sekundi i 633 milisekunde), dok je najveće povećanje uočeno testiranjem na simulatoru iPhone 8 (2 minute 36 sekundi i 720 milisekundi). Na brzinu pronalaska određenog elementa i na brzinu izvođenja određenih akcija, prilikom izvođenja testiranja, nije moguće utjecati, što zapravo objašnjava povećanja/smanjenja potrebnog vremena za izvođenje testiranja funkcionalnosti.

Testirana aplikacije, u sklopu ovog diplomskom radu, i dalje se razvija i poboljšava što će zasigurno dovesti do potrebe za unaprjeđenjem izrađenog projekta za automatizirano testiranje. Projekt će biti potrebno unaprijediti dodavanjem i implementacijom novih testnih slučajeva.

LITERATURA

[1] D. Knott, Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business, Pearson Education, Inc., Crawfordsville, Indiana, 2015.

[2] G. J. Myers, C. Sandler, T. Badgett: The Art of Software Testing, John Wiley & Sons, Inc., New Jersey, 2012.

[3] B. Hetzel, The Complete Guide to Software Testing, QED Information Sciences, Inc., Wellesley, Massachusetts, 1988.

[4] G. Martinović, Metode i tehnike testiranja programske podrške, prezentacija, Tipovi, razine i metode testiranja, FERIT, 2021.

[5] D. Graham, E. Van Veenendaal, I. Evans, R. Black, Foundations of Software Testing, Cengage Learning Emea, 2008.

[6] Daria R., A Step-By-Step Guide to Testing Healthcare Applications [online], RubyGarage, 5.5.2020., dostupno na : https://rubygarage.org/blog/testing-healthcare-application, [22.8.2023.]

[7] I. Vidović, Metode i tehnike testiranja programske podrške, prezentacija, Testing methods and test types, FERIT, 2019.

[8] Android Developers, Espresso [online], Android Developers, 07.03.2023, dostupno na: https://developer.android.com/training/testing/espresso, [22.8.2023.]

[9] A. R. Chowdhury, Automation Mobile Testing Frameworks [online], BrowserStack, 15.03.2023., dostupno na: https://www.browserstack.com/guide/mobile-application-testing-frameworks , [22.8.2023.]

[10] H. Rajora, Best Mobile App Testing Framework for Android and iOS Applications [online], LambdaTest, 22.12.2021., dostupno na: https://www.lambdatest.com/blog/best-mobile-app-testing-framework/, [22.8.2023.]

[11] Testsigma Technologies Inc., 11 Best Mobile Testing Tools [online], Testsigma Technologies Inc., dostupno na: https://testsigma.com/mobile-testing-tools, [22.8.2023.]

[12] RobotiumTech, User scenario testing for Android [online], GitHub, dostupno na: https://github.com/RobotiumTech/robotium, [22.8.2023.]

[13] Katalon, Top 10 Automated Mobile Testing Tools in 2023 | Latest Update [online], Katalon, 2023., dostupno na: https://katalon.com/resources-center/blog/mobile-testing-tools , [22.8.2023.]

[14] Apple Developer, XCTest [online], Apple, dostupno na: https://developer.apple.com/documentation/xctest, [23.8.2023.]

[15] N. Biswas, Comparing XCTest vs. XCUITest: How Do They Relate? [online], 6.12.2022., dostupno na: https://www.waldo.com/blog/xctest-vs-xcuitest, [23.8.2023.]

[16] Katalon, Katalon Studio [online], Katalon, dostupno na: https://katalon.com/katalon-studio, [23.8.2023.]

[17] OpenJS Foundation, Appium Documentation [online], OpenJS Foundation, dostupno na: https://appium.io/docs/en/2.0/ , [23.8.2023.]

[18] OpenJS Foundation, Overview of Appium [online], OpenJS Foundation, dostupno na: https://appium.io/docs/en/2.0/intro/, [23.8.2023.]

[19] OpenJS Foundation, Intro to Appium Clients [online]. OpenJS Foundation, dostupno na: https://appium.io/docs/en/2.0/intro/clients/, [23.8.2023.]

[20] Appium, Appium Desktop [online], GitHub, dostupno na: https://github.com/appium/appium-desktop#readme, [23.8.2023.]

[21] OpenJS Foundation, Server CLI Args [online], OpenJS Foundation, dostupno na: https://appium.io/docs/en/2.0/cli/args/ , [23.8.2023.]

[22] Appium, Appium Inspector [online], GitHub, dostupno na: https://github.com/appium/appium-inspector, [23.8.2023.]

[23] OpenJS Foundation, Capabilities [online], OpenJS Foundation, dostupno na: https://appium.io/docs/en/2.0/guides/caps/, [23.8.2023.]

[24] J. Ferguson Smart, BDD in Action: Behavior-Driven Development for the whole software lifecycle, Manning Publication, 2014.

[25] Cucumber, Introduction [online], SmartBear Software, dostupno na: https://cucumber.io/docs/guides/overview/, [28.8.2023.]

[26] A. Hellesoy, M. Wynne, The Cucumber Book: Behaviour-Driven Development for Testers and Developers, The Pragmatic Bookshelf, 2012.

[27] SmartBear Software, Zephyr Squale Cloud Document: Test Automation: [online], dostupno na: https://support.smartbear.com/zephyr-squad-cloud/docs/test-automation/index.html, [28.8.2023.]

[28] Atlassian, Welcome to Jira Software [online] Atlassian, dostupno na: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software, [28.8.2023.]

[29] Atlassian, Jira Software projects overview [online], Atlassian, dostupno na: https://www.atlassian.com/software/jira/guides/projects/overview#what-is-a-jira-project, [28.8.2023.]

[30] Atlassian, Jira Software boards introduction [online], Atlassian, dostupno na: https://www.atlassian.com/software/jira/guides/boards/overview#what-is-a-jira-board, [28.8.2023.]

[31] Atlassian, Jira Software issues overview [online], Atlassian, dostupno na:

https://www.atlassian.com/software/jira/guides/issues/overview#what-is-an-issue, [28.8.2023.]

[32] Cucumber, Gherkin References [online], SmartBear Software, dostupno na: https://cucumber.io/docs/gherkin/reference/, [28.8.2023.]

[33] Cucumber, Spoken languages [online], SmartBear Software, dostupno na: https://cucumber.io/docs/gherkin/reference/#spoken-languages, [28.8.2023.]

[34] Cucumber, Step Definitions [online], SmartBear Software, dostupno na: https://cucumber.io/docs/cucumber/step-definitions/?lang=java, [28.8.2023.]

[35] Software Testing Help, TestNG Example: How To Create And Use TestNG.Xml File [online], Software Testing Help, 20.6.2023., dostupno na: https://www.softwaretestinghelp.com/testng-example-to-create-testng-xml/, [28.8.2023.]

SAŽETAK

U ovom diplomskom radu je, korištenjem Appium alata, automatizirano testirana zdravstvena aplikacija za vođenje dnevnika o glavoboljama. U teorijskom dijelu rada opisano je testiranje mobilnih aplikacija i klasificirani su postojeći alati za automatizirano testiranje mobilnih aplikacija od kojih su pojedini i opisani. U praktičnom dijelu rada odrađen je testni proces u kojem je odrađeno testno planiranje, analiza zahtjeva, dizajniranje i implementacija testnih slučajeva, funkcionalno testiranje i izvještavanje. Za potrebe izvođenja automatiziranog testiranja korišteno je razvojno okruženje IntelliJ IDE, Appium, Cucumber i TestNG alati. Implementacija automatiziranih testova realizirana je korištenjem Java programskog jezika te je za potrebe upravljanja testiranjem korišten Zephyr Squad alat. Automatizirano testiranje izvedeno je korištenjem lokalnog i udaljenog poslužitelja. Nakon izvedenog testiranja, rezultati su analizirani te je obavljena usporedba potrebnog vremena za izvođenje automatiziranog testiranja prilikom korištenja lokalnog poslužitelja i udaljenog poslužitelja.

Ključne riječi: Appium, automatizirano testiranje, mobilna aplikacija, Cucumber, Zephyr

ABSTRACT

Appium framework for automation testing of mobile applications

In this Master's thesis, the Appium tool was used for automated testing of a healthcare diary application that tracks headaches. In theoretical part of this thesis, the testing of mobile applications is described and existing tools for automated testing of mobile applications are classified, some of which are also described. In the practical part of this thesis, the testing process was carried out, involving test planning, requirements analysis, design and implementation of test cases, functional testing, and reporting. The IntelliJ IDE development environment, along with Appium, Cucumber, and TestNG tools, were used to perform automated testing. The implementation of automated tests was realized using the Java programming language, and Zephyr Squad tool was used for the purposes of test management. Automated testing was executed using both local and remote servers. After performed testing, the results were analyzed, and a comparison of the time required to perform automated testing using local server and remote server was made.

Keywords: Appium, automated testing, mobile application, Cucumber, Zephyr